Aus dem Institut für Radiologie
der Medizinischen Fakultät der Charité
Universitätsmedizin Berlin

Dissertation

softMip

Entwicklung eines neuen Projektionsverfahrens für die digitale Schnittbildgebung und Evaluation anhand von Ultra-Low-Dose-CT-Aufnahmen zur Harnwegskonkrementsuche

zur Erlangung des akademischen Grades
Doctor medicinae
(Dr. med.)

vorgelegt der Medizinischen Fakultät der Charité
Universitätsmedizin Berlin

von
Henning Meyer
aus Berlin

Gutachter:

1. Prof. Dr. med. B. Hamm
2. Prof. Dr. med. D. Schnorr
3. Prof. Dr. med. S. Feuerbach

Datum der Promotion: 10. Dezember 2005

Abstract

Two projection algorithms exist in the cross sectional imaging: Average projection, which features good suppression of image noise, however the edge sharpness diminishes. In maximum-intensity-projection (MIP), images show good edge sharpness, but also amplified image noise. Ultra-Low-Dose (ULD)-CT has very low radiation exposure, but suffers from high image noise. MIP images of ULDCT amplify to image noise and are therefore unusable for the clinical routine. We developed a synthesis of both algorithms which tries to unite the respective advantages. The resulting softMip represents an projection algorithm which is a mathematic combination of both MIP and average projection. It was implemented in C++ and installed on a workstation. Depending on the settings softMip can result in any graduation between MIP and average projection. In a phantom trial on seven different CT-devices, image noise and edge sharpness of the three projection algorithms (MIP, Average and softMip) were aquired. Furthermore image quality of the transition from Average to MIP by means of softMip was compared to the image quality of alpha blending average projection and MIP. In a clinical trial we investigated softMip compared to MIP and average projection in ULDCT examinations of 31 patients with suspected calcifications in the urinary tract. A softMip reading was compared to a reading with the combination of MIP and Average and a reading with the originals thin slices as reference standard. In each run, amount, size and localization of the calcifications were documented. For each projection algorithm, the subjective criteria image noise, contrast, contour sharpness, artifacts and overall image impression were documented. Finally the reader was asked to select six images that document the relevant findings best. softMip showed a less image noise than a MIP ($p<0.0005$) and higher edge sharpness than Average ($p<0{,}0005$). softMip transition from average projection to MIP had a better ratio of edge sharpness and image noise than alpha blending technique ($p<0.0005$). No significant differences in the overall image quality between Average and softMip could be shown.

The sensitivity of the detection of kidney calcifications is higher in softMip than the sensitivity with MIP and average projection combined (0.81 vs. 0.60; $p<0.0005$). The specificity of the dection in softMip is only slightly lower than the specificity in MIP and average projection combined (0.74 vs. 0.95, $p=0.13$). Regarding ureteral calcification, no significant differences in sensitivity and specificity was observed. 67% of all documented images were selected from softMip, 33% out of average projection and thin slices. softMip can improve the detection of urinary calcifications in ultra low dose CT and therefore represents a promising projection algorithm for ultra low dose CT.

Keywords:
computed tomographie, projection, image noise, algorithm

Zusammenfassung

Bisher sind in der Schnittbildgebung zwei Projektionsverfahren üblich: Die Average-Projektion, die das Bildrauschen gut unterdrückt, jedoch die Kantenschärfe verringert und die Maximum-Intensitäts-Projektion (MIP), deren Bilder sich durch eine gute Kantenschärfe auszeichnen, gleichzeitig aber ein höheres Bildrauschen aufweisen. Ultra-Low-Dose (ULD)-CT-Aufnahmen weisen ein hohes Bildrauschen auf. MIP-Bilder von ULDCT-Aufnahmen verstärken dieses Bildrauschen und sind deshalb für die klinische Routine ungeeignet. Daher wurde in der vorliegenden Arbeit eine Synthese beider Verfahren, mit dem Ziel die jeweiligen Vorteile zu vereinen, entwickelt. Das daraus resultierende softMip-Verfahren wurde implementiert und auf einer Workstation eingerichtet. In einer Phantom-Studie mit sieben verschiedenen CT-Geräten wurden das Bildrauschen und die Kantenschärfe der drei Projektionsverfahren (MIP, Average und softMip) ermittelt. Anschließend wurden die drei Verfahren in einer klinischen Studie bei der Suche nach Harnwegskonkrementen in 31 ULD-CT-Abdomenaufnahmen eingesetzt. Ein erfahrener CT-Radiologie führte jeweils eine Befundung mit Bildern der softMip, mit der Kombination aus MIP und Average und mit den originalen Dünnschichten durch. In jedem Durchgang wurden Anzahl, Größe und Lokalisation der Harnwegskonkremente erfasst. Für jedes Projektionsverfahren wurden darüber hinaus die subjektiven Kriterien Bildrauschen, Kontrast, Konturschärfe, Artefakte und Bildeindruck erhoben. Zum Abschluss wählte der Radiologe maximal sechs Bilder aus den gegebenen Verfahren aus, die den Befund am treffendsten dokumentieren.

Im Ergebnis des Phantomversuches zeigte sich, dass softMip ein geringeres Bildrauschen als MIP aufweist ($p<0{,}0005$) und die Kantenschärfe höher ist als die der Averageprojektion ($p<0{,}0005$). In der klinischen Studie konnten keine signifikanten Unterschiede in der Gesamtbildqualität zwischen Average und softMip nachgewiesen werden. Die Sensitivität gegenüber Nierenkonkrementen in der softMip ist höher als die der MIP-Average-Kombination (0,81

vs. 0,60; p<0,0005). Die Spezifität in der softMip ist nur tendenziell geringer (0,74 vs. 0,95, p=0,13). Bezüglich Ureterkonkrementen konnten keine signifikanten Unterschiede in Sensitivität und Spezifität nachgewiesen werden. 67% aller Bilder der Befunddokumentation wurden aus softMip-Serien gewählt, 33% aus Average-Serien und den Dünnschichten.

So konnte gezeigt werden, dass die softMip ein vielversprechendes Projektionsverfahren für die digitale Schnittbildgebung insbesondere im ULD-Bereich darstellt.

Schlagwörter:
Computertomographie, Projektion, Bildrauschen, Algorithmus

Danksagung

Meinem Betreuer, Dr. Rogalla, bin ich zu großem Dank verpflichtet - für die Überlassung des Themas, die abendlichen und nächtlichen Befundungssitzungen und die vielen anregenden Diskussionen und Hinweise, die mich zum weiteren selbstständigen Arbeiten animierten.

Herrn Professor Hamm danke ich dafür, dass er die Voraussetzungen für diese Arbeit an seiner Klinik schuf.

Ein großer Dank gilt Dr. C. Klüner für die Überlassung der Ultra-Low-Dose-Studiendaten, ohne die diese Arbeit so nicht möglich gewesen wäre.

Herrn Juran bin ich besonders für die Beratung in Fragen zur Physik und Technik der CT dankbar.

Bei den MRAs des Instituts bedanke ich mich für die Hilfe bei der Durchführung der Phantomuntersuchungen.

Weiterhin bedanken möchte ich mich bei Marc und Franzi für Rat und Hilfe bei statistischen Fragen.

Meine Freundin Stephanie und ihr Sohn Timothy nahmen so oft Rücksicht auf mich und meine Promotion und waren für mich da, wenn alles mal wieder nicht klappen wollte – vielen Dank dafür!

Meinen Eltern danke ich für das Vertrauen, welches sie mir entgegenbrachten und für die große Unterstützung und Hilfe, die sie mir während des Studiums und auch der anschließenden Promotion gaben. Vielen Dank – ich bin froh, Euch als Eltern zu haben.

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Ziel der Arbeit 2

2 Grundlagen **3**
2.1 Urolithiasis 3
2.2 Computertomographie 3
2.2.1 (Ultra)-Niedrigdosis-CT 5
2.3 Nachverarbeitungsverfahren 6
2.3.1 Nachverarbeitung von ULDCT-Aufnahmen 8

3 Material und Methoden **9**
3.1 softMip 9
3.2 DicomProjector 10
3.3 Phantomversuch 11
3.3.1 Phantom 11
3.3.2 Datenaquisition 11
3.3.3 GradientFinder 12
3.3.4 MIP-Average-Blending 15
3.4 Klinische Studie 16
3.4.1 Aquisition der Patienten und der Bilddaten 16
3.4.2 Befundung 17
3.4.3 Auswertung 18
3.4.3.1 Konkremente/StoneSorter 18
3.4.3.2 Stauung 21
3.5 Statistik 21

4 Ergebnisse **22**
4.1 softMip-Server . . . 22
4.2 Wichtungsfunktionen . . . 22
4.3 Phantomversuch . . . 24
4.3.1 Blending-Vergleich . . . 26
4.4 Klinische Studie . . . 27
4.4.1 Subjektive Bildqualität . . . 30
4.4.2 Konkremente . . . 30
4.4.2.1 Diagnostische Aussagekraft (Dünnschichten als Referenz) . . . 32
4.4.2.2 Diagnostische Aussagekraft (Konsensus als Referenz) . . . 34
4.4.3 Stauung . . . 35
4.4.3.1 Diagnostische Aussagekraft (Dünnschichten als Referenz) . . . 35
4.4.3.2 Diagnostische Aussagekraft (Konsensus als Referenz) . . . 36
4.4.4 Dokumentation . . . 36

5 Diskussion **39**
5.1 Computertomographie und Strahlenexposition . . . 39
5.2 softMip-Ergebnisse . . . 40
5.3 Vergleich mit ähnlichen Projekten . . . 42
5.3.1 Rauschminderung in den axialen Schichten . . . 42
5.3.2 Rauschminderung vor der Rekonstruktion . . . 43
5.3.3 Auflösung und Bildrauschen . . . 43
5.3.4 Anwendungen der MIP . . . 43
5.4 Limitierungen . . . 44
5.4.1 Clusterfehler . . . 44
5.4.2 Toleranz bei Konkrementgröße und -lokalisation . . . 44
5.4.3 Korrigierte Referenz . . . 45
5.4.4 Tools und Scripte . . . 45
5.5 Ausblick . . . 46

A DicomProjector **55**
A.1 C++ - Header 55
A.2 C++ - Body 58

B GradientFinder **74**

C StoneSorter **85**

Abbildungsverzeichnis

2.1 Funktionsweise von MIP und Average 7

3.1 Funktionsweise von softMip 10
3.2 Konstanzprüfphantom 11
3.3 GradientFinder 13
3.4 Ablaufdiagramm von GradientFinder 14
3.5 Normalisierung des Rauschens 15
3.6 Befundungsbogen des softMip-Durchgangs 17
3.7 Befundungsbogen des Mip-Average-Durchgangs 19
3.8 Ablaufdiagramm von StoneSorter 20

4.1 Kantenschärfe bei den einzelnen Geräten 24
4.2 Rauschen bei den einzelnen Geräten 25
4.3 Kantenschärfe und Rauschen im Phantomversuch 25
4.4 Blendingkurven von softMip und MIP-Average mit 25% und 75%-Quantil 26
4.5 Fläche unter der Blendingkurve für alle Geräte 27
4.6 prävesikales Konkrement konventionell 28
4.7 prävesikales Konkrement in den 50 mm CT-Projektionen . . . 29
4.8 Nierenkonkrement in den 10 mm CT-Projektionen 29
4.9 Nierenkonkrement vergrößert 30
4.10 Bewertung der Bildqualität (Schichtdicke 10 mm) 31
4.11 Bewertung der Bildqualität (Schichtdicke 50 mm) 32
4.12 relative Häufigkeiten der Projektionen in der Dokumentation . 37

Tabellenverzeichnis

3.1 für den Phantomversuch benutzte CT-Geräte 12

4.1 eingesetzte Wichtungsfunktionen 23
4.2 Anzahl der gemessenen Kanten 23
4.3 Unterschiede in der subjektiven Bildqualität (10 mm) 31
4.4 Unterschiede in der subjektiven Bildqualität (50 mm) 31
4.5 kombinierte Kreuztabelle für Nierenkonkremente mit Dünnschichtreferenz 33
4.6 Kreuztabellen bezüglich Nierenkonkrementen mit Dünnschichtreferenz 33
4.7 kombinierte Kreuztabelle für Ureterkonkremente mit Dünnschichtreferenz 33
4.8 kombinierte Kreuztabelle für Nierenkonkremente mit Konsensusreferenz 34
4.9 Kreuztabellen bezüglich Nierenkonkrementen mit Konsensusreferenz 34
4.10 kombinierte Kreuztabelle für Ureterkonkremente mit Konsensusreferenz 35
4.11 kombinierte Kreuztabelle für Nierenstauung mit Dünnschichtreferenz 35
4.12 Kreuztabellen bezüglich Nierenstauung mit Dünnschichtreferenz 36
4.13 kombinierte Kreuztabelle für Nierenstauung mit Konsensusreferenz 37

Kapitel 1

Einleitung

Der akute Flankenschmerz stellt eine klinische Symptomatik dar, deren häufigste Ursache die Urolithiasis ist. Zur Routinediagnostik bei Verdacht auf Urolithiasis gehört sowohl der Ultraschall als auch die native Abdomenübersicht. Die Computertomographie (CT) ist dieser Diagnostik in der Sensitivität überlegen [47], konnte sich jedoch in der klinischen Routine aufgrund der erhöhten Stahlenexposition des Patienten nicht durchsetzen. Jüngere Studien zeigen jedoch, dass die Ultra-Niedrig-Dosis-CT (ULDCT) eine Sensitivität aufweist, die mit der der konventionellen CT vergleichbar ist [30]. Die Strahlenexposition weicht dabei nicht relevant von der einer nativen Abdomenübersicht ab [48]. Allerdings ist das Bildrauschen der ULDCT gegenüber der konventionellen CT durch eine Einsparung von 95% der Röntgendosis deutlich erhöht. Die Mehrschicht-Spiral-CT (MSCT) ermöglicht eine Untersuchung des Abdomens mit einer Schichtdicke von 1 mm in einem Atemstillstand. So ergeben sich 500-600 Bilder pro Patient, deren Informationen die Aussagekraft des Röntgens und des Ultraschalls übersteigen.

Eine solche Bilderflut und das erhöhte Bildrauschen machen zur Diagnosestellung eine Nachverarbeitung der ULDCT-Daten notwendig. Bisher etablierte Methoden für die CT sind die Average-Projektion oder die Maximum-Intensitäts-Projektion (MIP), welche aus vielen dünnschichtigen Bildern wenige prägnante Bilder generieren. Die Average-Projektion vermag durch mathematische Mittelwertbildung mehrerer Dünnschichten das Bildrauschen in

der resultierenden Dickschicht wirksam zu unterdrücken. Eine Stärke der MIP ist das Hervorheben röntgendichter Strukturen, wie kontrastierte Gefäße oder Verkalkungen. Beide Verfahren sind für die Verarbeitung von ULDCT-Daten zur Konkrementsuche nicht ausreichend, da einerseits kleinere Konkremente in dickschichtigen Average-Projektionen verlorengehen und andererseits das hohe Rauschen der ULDCT durch die MIP weiter verstärkt wird.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist die Entwicklung und Evaluation eines Projektionsalgorithmus für die MSCT, welcher die Bildqualität, ähnlich der Average-Projektion, durch Verminderung des Rauschens steigert und dabei gleichzeitig zu einer verbesserten Erkennbarkeit pathologischer Bildmerkmale, ähnlich der MIP, führt. Die Evaluation soll anhand objektiver Kriterien durch eine Phantomstudie mit Bestimmung der Kantenschärfe und des Bildrauschens, sowie anhand einer klinischen Anwendung geschehen.

Kapitel 2

Grundlagen

2.1 Urolithiasis

An Nierensteinen leiden bis zu 5% der westlichen Bevölkerung, wobei sich bei erhöhtem Lebensstandard auch ein erhöhtes Risiko für eine Nephrolithiasis ergibt [42]. Männer sind doppelt so häufig betroffen wie Frauen. Der Altersgipfel bei Männern liegt bei 30 Jahren, bei Frauen liegt ein zweigipfliges Geschehen mit Gipfeln bei 35 und 55 Jahren vor. 75-85% aller Nierensteine sind Kalziumsteine [2]. Die häufigste Ursache ist mit 50-55% die idiopathische Hyperkalziurie, gefolgt von der Hyperurikosurie mit 20% Häufigkeit. Steinerkrankungen können längere Zeit asymptomatisch bleiben. Sie sind neben malignen und benignen Neoplasien, Nierenzysten und der Urogenitaltuberkulose der häufigste Grund für eine Hämaturie. Zerbricht ein Stein, so kann er beim Eintritt in den Ureter oder beim Verschluss des Nierenbeckens eine Obstruktion verursachen, die zu Schmerzen und starken Koliken führen kann.

2.2 Computertomographie

Die Computertomographie (CT) ist eine digitale Röntgenmethode zur Gewinnung horizontaler Schichtbilder. Setzt man diese Schichten zusammen, so erhöht man ein dreidimensionales Volumen. Mittels moderner Mehrschicht-CT(MSCT)-Geräte ist es möglich, eine große Menge hochauflösender Schnitt-

bilder innerhalb kürzester Zeit zu gewinnen [22, 41, 45]. So können heute bei CT-Untersuchungen des Abdomens innerhalb einer Atemanhaltephase ca. 500 Schnittbilder mit 0,8 mm Schichtabstand gewonnen werden. Durch die gesteigerte Auflösung, insbesondere in der Längsachse des Patienten, ist es nun auch möglich, echte 3D-Volumendatensätze zu aquirieren. ähnlich wie 2D-Bilder aus einzelnen Bildpunkten – den Pixeln – aufgebaut sind, ist ein Volumendatensatz aus Volumenpunkten – den Voxeln – aufgebaut. Der Wert eines Voxels entspricht der Schwächung der Röntgenstrahlung an diesem Ort. Die Einheit dieser Schwächung ist die Hounsfield Unit (HU).

Die CT ist in der Diagnostik des akuten Flankenschmerzes aufgrund des hohen diagnostischen Informationsgehalts dem Ultraschall und der nativen Abdomenbersicht in der Sensitivität überlegen [47]. Der hohen diagnostischen Aussagekraft der CT steht allerdings die – gegenber der Abdomenbersicht deutlich erhöhte – Strahlenexposition gegenüber. In Europa und den USA machen CT-Untersuchungen ca. 3-5% aller radiologischen Untersuchungen aus. Gleichwohl beträgt aber der Anteil der CT an der gesamten durch Ärzte verursachten diagnostischen Strahlenexposition 35-45% [11, 20, 36, 44]. Mit der Einführung der 4-Zeilen-CT wurde das Problem noch verstärkt, da jede der vier Detektorzeilen die gleiche Strahlungsintensität benötigt. Um dies zu gewährleisten, muss die Strahlungskollimation so gewählt werden, dass das Strahlungsprofil erst außerhalb der Detektorzeilen abfällt. So ergibt sich ungenutze Strahlung, die nicht zur Messung beiträgt, jedoch die Exposition des Patienten erhöht [5, 25]. Dieser Effekt ist umso stärker, je kleiner die Schichtkollimation ist. Mit größerer Kollimation und steigender Detektorzeilenzahl nimmt der Effekt ab. Zur Verringerung der Strahlendosis der CT wurden verschiedene Verfahren entwickelt [24]. Eine Möglichkeit ist die Anpassung des Verlaufes des Röhrenstromes und dadurch der Strahlendosis an die Dicke des zu untersuchenden Körperabschnitts [12, 13, 18, 23]. Jedes Verfahren, welches das Bildrauschen einer CT-Aufnahme mindert, kann prinzipiell auch dazu benutzt werden, die Dosis bei gleichbleibendem Rauschen zu mindern. Vielfältige entsprechende Ansätze wurden bereits vorgeschlagen [3, 14, 27, 52].

2.2.1 (Ultra)-Niedrigdosis-CT

Um der erhöhten Strahlenexposition der CT gegenüberzutreten ist als erster Schritt – wie bei jeder anderen Untersuchung auch – die klinische Indikation durch den Radiologen genau zu prüfen. Bei bestimmten Fragestellungen – wie z.B. der Lungenrundherderkennung – bietet sich der Einsatz der Niedrigdosis-CT (Low-dose CT) an. Durch die verringerte Röntgendosis sinkt auch die Zahl der Röntgenquanten, die den Detektor erreichen, und die Messungenauigkeit steigt. Dies wirkt sich als sichtbares Bildrauschen aus, welches mit sinkender Dosis steigt – und umgekehrt. Das bei Low-Dose-CT-Aufnahmen deutlich erhöhte Bildrauschen erschwert besonders die Wahrnehmung von Strukturen, die nur einen geringen Kontrast zur Umgebung haben. Strukturen mit hohem Kontrast werden jedoch auch bei erhöhtem Rauschen noch gut wahrgenommen. Strukturen mit hohem Bildkontrast in der CT sind beispielsweise die Nasennebenhöhlen, wo Luft auf das Weichteilgewebe der Nasenschleimhaut trifft, oder Knochen, deren Dichte wesentlich höher als die des umgebenden Weichteilgewebes ist. Die Low-Dose-Technik wird im klinischen Alltag bereits für Thorax-CT-Untersuchungen benutzt. Lungenrundherde oder pneumonische Infiltrate lassen sich auch bei verminderter Dosis gut von normalem lufthaltigen Lungengewebe abgrenzen. Einschränkungen gibt es aber beispielsweise bei der Beurteilung der mediastinalen Weichteile [8, 31, 40, 43]. Auch für die virtuelle Koloskopie wird die Low-Dose-Technik benutzt, da es sich bei dem für diese Untersuchung relevanten Übergang von Darmwand zu Luft um einen Hochkontrast-Übergang handelt [7, 19, 54].

In jüngster Zeit hat sich der Trend zur Dosisminimierung noch einen Schritt weiterentwickelt. Rogalla, Klüner und Taupitz berichten über den Einsatz der Ultra-Niedrigdosis-CT (Ultra-low-dose CT/ULDCT) zur Konkrementsuche in Nieren und ableitenden Harnwegen [48]. Die Strahlenexposition einer ULDCT-Abdomen-Aufnahme entspricht mit durchschnittlich 0,6 mSv der einer einzelnen Röntgen-Abdomenaufnahme. Im Vergleich dazu beträgt die Dosis einer Normaldosis-CT des Abdomes 5-10 mSv. Es konnte gezeigt werden, dass auch die ULDCT signifikant bessere Ergebnisse in der Steindiagnostik liefert als der Ultraschall.

2.3 Nachverarbeitungsverfahren

Um der Datenmenge Herr zu werden, benutzt man heute bei der Auswertung von CT-Untersuchungen Computer-Workstations, auf welchen man die gewonnenen Daten komfortabel betrachten und nachverarbeiten kann [51]. Je nach eingesetzter Software und klinischer Fragestellung bieten sich verschiedene Nachverarbeitungen an. Schon zu Beginn der Computernutzung zeichneten sich klare Vorteile ab: Einerseits kann man am Monitor die Bilder größer als auf den Filmen darstellen, andererseits ist es möglich, Window-Level und Window-Width frei zu wählen. Man kann so Helligkeit und Kontrast der angezeigten Bilder im Nachhinein regeln und beispielsweise nacheinder die Lungenstruktur, das Weichteilgewebe und die Knochen beurteilen. Mit der Einführung der Spiral-CT und der daraus resultierenden höheren Z-Achsen-Auflösung wurde es mittels der multiplanaren Reformatierung (MPR) möglich, aus den axialen Schichten neue Ansichten in koronaler, sagittaler oder beliebiger Orientierung zu erzeugen. Aus mehreren dünnen Schichten kann man mit der Average-Projektion eine einzige Schicht errechnen, die früheren Dickschichtaufnahmen gleichkommt. Da die Average-Projektion ein 2D-Bild aus dem Mittelwert mehrerer Schichten eines Volumens berechnet, wird in diesen Bildern das Rauschen gut reduziert. Allerdings gehen dabei auch die Bilddetails und die Kontraste verloren. Insbesondere bei dickschichtigen Daten kann es vorkommen, dass eine Struktur – z.B. ein kontrastiertes Gefäß – nur zum Teil in einem Voxel der zu betrachtenden Schicht liegt. Der HU-Wert dieses Voxels spiegelt deshalb auch nur zum Teil den des Gefäßes wider. Diesem Partialvolumeneffekt muss bei der Interpretation des Bildes Rechnung getragen werden [57].

Der Average-Projektion ähnelt die Maximum-Intensitäts-Projektion. Diese wurde zuerst von Laub und Kaiser sowie Keller et al. zur Nachverarbeitung von MR-Angiographie-Untersuchungen vorgeschlagen [28, 33]. Die MIP ist eine Zusammensetzung der jeweils hellsten Voxel eines Volumens. Dadurch weisen MIP-Bilder gute Kontraste auf, jedoch macht sich ein erhöhtes Bildrauschen sehr stark bemerkbar. Bei der Average-Projektion und der MIP wird aus einer beliebig dicken dreidimensionalen Volumenscheibe

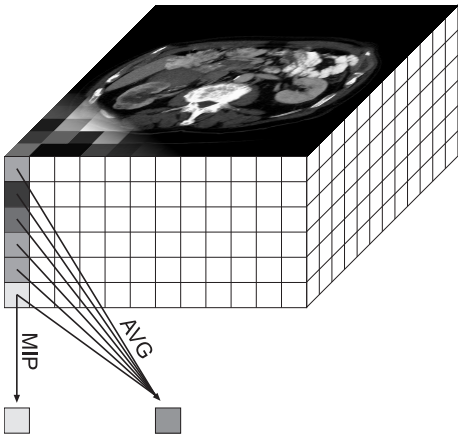


Abbildung 2.1: Funktionsweise von MIP und Average

ein zweidimensionales Bild berechnet. Dabei entspricht jedem Bildpunkt (Pixel) des resultierenden Bildes ein Stapel von Voxeln der Volumenscheibe. Abbildung 2.1 zeigt die grundlegenden Verfahren der MIP und der Average-Projektion. Bei der MIP wird der jeweils hellste Punkt des Voxelstapels zum entsprechenden Pixel des Bildes. Bei der AVG ergibt sich das Pixel als arithmetisches Mittel des Voxelstapels. Die Berechung von MIP-Schichten ist aufwendig, konnte aber so beschleunigt werden, dass sie sich gut in den klinischen Alltag integrieren lässt [38]. Ein wichtiges Anwendungsgebiet der MIP ist die CT-Angiographie (CTA) [34, 37, 46]. Bei der Beurteilung von CT-Thoraxaufnahmen sind die MIP und die MPR hilfreich [3, 9, 39].

Andere Verfahren, wie Volumerendering (VR) oder Shaded Surface Display (SSD), erfordern zunächst eine Segmentierung bzw. Klassifizierung der Daten. Dabei wird vom Benutzer festgelegt, welche Anteile des Datensatzes visualisiert werden sollen, und welche Anteile nicht dargestellt werden. Als Resultat erhöht man teilweise sehr realistische dreidimensionale Darstellungen der untersuchten Organe. Dies ist sehr hilfreich, um z.B. einem Traumatologen eine Fraktur und die Stellung der einzelnen Fragmente zu veranschaulichen. Die Information über die innere Struktur der Organe geht dabei jedoch weitestgehend verloren. Daher sind diese Verfahren für viele Fragestellungen nicht Mittel der Wahl.

2.3.1 Nachverarbeitung von ULDCT-Aufnahmen

Zur Steinsuche eignet sich prinzipiell die MIP sehr gut, da sie röntgendichte Objekte – etwa Konkremente – besonders gut darstellt. ULDCT-Aufnahmen weisen allerdings ein sehr hohes Bildrauschen auf. Die MIP verstärkt dieses Rauschen sogar, da sich MIP-Bilder selektiv aus allen Voxeln mit hohen HU-Werten – und so aus allen besonders "rauschbelasteten" Voxeln – zusammensetzen. Um das Rauschen zu verringern, bietet sich die Average-Projektion an. Diese kann jedoch durch den wirkenden Partialvolumeneffekt das Auffinden kleiner Konkremente vereiteln.

Wünschenswert ist ein Verfahren, welches das Rauschen – ähnlich der Average-Projektion – wirksam unterdrckt und dabei dennoch – wie die MIP – kontrastreiche Strukturen und Pathologien hervorhebt.

Kapitel 3

Material und Methoden

3.1 softMip

Ein Ansatz, in CT-Aufnahmen das Bildrauschen zu unterdrücken und Kontraste anzuheben, ist das für diese Arbeit entwickelte Projektionsverfahren softMip. An dieser Stelle soll der grundlegende Algorithmus der softMip erläutert werden. Es handelt sich dabei um ein nichtlineares Projektionsverfahren. ähnlich wie bei der MIP und der Average-Projektion wird ein Bildpunkt der Projektion aus den HU-Werten entlang des Sichtstrahls dieses Bildpunktes berechnet. Dazu werden nur die Werte in die Berechnung mit einbezogen, welche innerhalb der gewählten Schicht liegen. Diese Werte bilden das Profil P des Bildpunktes p. Die Länge l von P ist abhängig von der Schichtdicke s und von der Projektionsgeometrie – bei paralleler Projektion ist $l = s$, bei perspektivischer Projektion ist $l \geq s$. Fr die softMip entscheidend ist es nun, das Profil P aufsteigend zu sortieren. Man erhöht P^s mit $P^s(0) = \min(P)$ und $P^s(l) = \max(P)$. Der Bildpunkt p ergibt sich nun aus dem normalisierten, gewichteten Integral des sortierten Profils:

$$p = \frac{1}{\int_0^1 f_w(x)dx} \cdot \int_0^l f_w\left(\frac{x}{l}\right) \cdot P^s(x)dx$$

Dabei ist $f_w(x)$ mit $0 \leq x \leq 1$ die Wichtungsfunktion der softMip. Das Ergebnis der softMip hängt somit sehr stark von der Wichtungsfunktion f_w

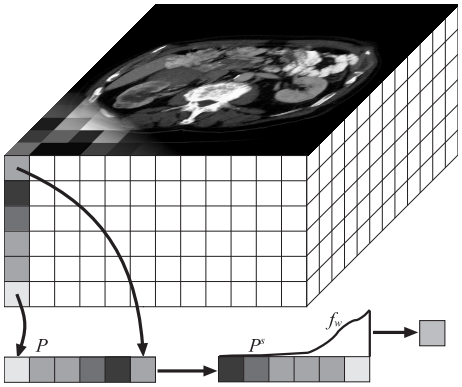


Abbildung 3.1: Funktionsweise von softMip

ab. Beispielsweise lässt sich mit $f_w^{\text{MIP}}(x) = \left\{ \begin{array}{l} 1 \text{ ;für } x > 0,999 \\ 0 \text{ ;sonst} \end{array} \right\}$ eine MIP erzeugen. $f_w^{\text{Average}} = 1$ hingegen entspricht der Average-Projektion. Eine softMip mit Rauschunterdrckung und MIP-Charakteristik stellt $f_w(x) = x^4$ dar.

3.2 DicomProjector

Der softMip-Algorithmus wurde in Form des Programm DicomProjector implementiert. Dazu wurde die Software Visual Studio.NET[1] und die Programmiersprache C++ genutzt. Zum Einlesen der CT-Bilddaten sowie zum Schreiben der erzeugten Projektionsdaten im Dicom-Format[2] wurde das Dicom-Toolkit DCMTK-3.2.5[3] verwendet. Fr die grafische Oberfläche wurde auf das Fox-Toolkit-1.0.26[4] zurückgegriffen. Im Anhang A sind die wesentlichen Abschnitte des Programmes abgedruckt.

[1]Microsoft Corporation, Redmond, USA (http://msdn.microsoft.com/vstudio)
[2]Digital Imaging and Communications in Medicine (http://medical.nema.org)
[3]Kuratorium OFFIS e.V., Oldenburg, Germany (http://dicom.offis.de)
[4]Fox-Toolkit, Jeroen van der Zijp (http://www.fox-toolkit.org)

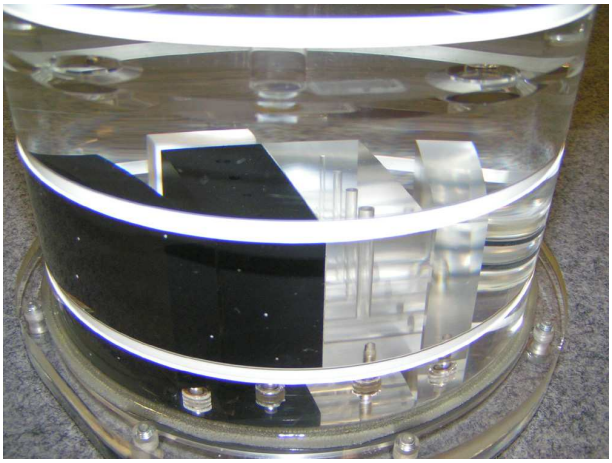


Abbildung 3.2: Konstanzprüfphantom

3.3 Phantomversuch

Um die mit softMip erzeugten Bilder objektiv beurteilen zu können, wurde ein Phantomversuch durchgeführt.

3.3.1 Phantom

Als Phantom wurde ein Konstanzprüfphantom[5] der Siemens Somatom DR-Serie benutzt (Abbildung 3.2). Es besteht aus einem mit Wasser gefüllten Plexiglaszylinder, in dem mehrere Quader, bestehend aus Teflon, Plexiglas und Polyethylen, befestigt sind. Dies ermöglicht sowohl die Erfassung von hohen Kontrasten zwischen Wasser mit 0 HU und Teflon mit $\sim$ 990 HU, als auch mittleren Kontrasten zwischen Wasser und Plexiglas mit $\sim$ 120 HU. Mit Polyethylen (~ -90 HU) waren auch Messungen mit negativen Kontrasten möglich.

3.3.2 Datenaquisition

Es wurden Messungen an diversen CT-Geräten sowie an einem Elektronenstrahl-CT (EBT) [4] durchgeführt, um sicherzustellen, dass die gefundenen

[5]Siemens Medical Solutions, Siemens AG, München

Gerätename	Technologie
Aquillion 4[6]	4-Zeilen-Spiral-CT
Aquillion 16[6]	16-Zeilen-Spiral-CT
Aquillion 64[6]	64-Zeilen-Spiral-CT
Tomoscan AV/EU[7]	Einzeilen-Spiral-CT
PQ5000[7]	Einzeilen-Spiral-CT
Brilliance 16[7]	16-Zeilen-Spiral-CT
Somatom Sensation 16[8]	16 Zeilen-Spiral-CT
EBT C-150XP[9]	Elektonenstrahl-CT

Tabelle 3.1: für den Phantomversuch benutzte CT-Geräte

Daten nicht von einem speziellen Gerätetyp abhängen. Im Einzelnen wurden die in Tabelle 3.1 aufgeführten CT-Geräte verwendet. Es wurde ein einheitliches Scanprotokoll in Niedrigdosis-Technik verwendet (1 mm Schichtdicke, 3 mAs effektiv, 120 kV, Pitch 1,43), sofern dies auf dem jeweiligen Gerät möglich war. Andernfalls wurden die Parameter entsprechend den Limitierungen des jeweiligen Gerätes angepasst. Für jeden Phantomscan wurden sowohl koronale und sagittale Projektionen mit 20 mm Schichtdicke in Average-Projektion, MIP und softMip berechnet.

3.3.3 GradientFinder

In den berechneten Projektionen sollten die Übergänge von Wasser zu den jeweiligen Quadern untersucht werden. Ziel war es, entlang dieser Übergänge sowohl die Stärke des Gradienten, der gut mit dem Bildkontrast korreliert, als auch das Bildrauschen, also die Standardabweichung über dem Gradienten, zu bestimmen. Um diese Messungen möglichst genau und objektiv durchzuführen, wurde ein Auswertungsprogramm (GradientFinder) in der Programmiersprache Ruby[10] programmiert. Dieses Programm las je eine zusammengehörige Average-, MIP- und softMip-Projektion ein, ermöglichte die

[6] Toshiba Medical Systems, Nasu, Japan
[7] Philips Medizin Systeme GmbH, Hamburg
[8] Siemens Medical Solutions, Erlangen
[9] GE Imatron, GE Medical Systems, San Francisco, USA
[10] http://www.ruby-lang.org

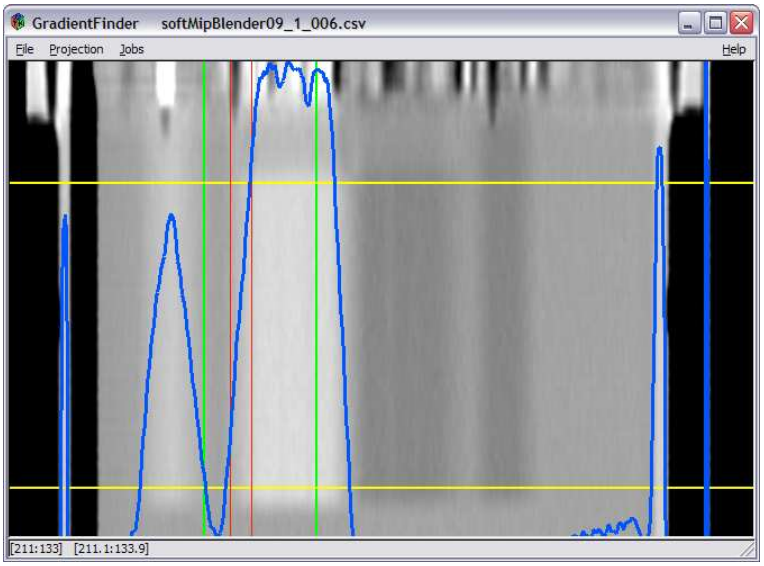


Abbildung 3.3: GradientFinder

Markierung aller relevanten Übergänge und übernahm dann die erforderlichen Messungen. Abbildung 3.3 zeigt ein Bildschirmfoto des Programmes. Zu sehen ist ein Schnitt durch das Phantom mit dem gemittelten Profil innerhalb des gewählten Ausschnitts. Abbildung 3.4 zeigt schematisch die Funktionsweise von GradientFinder.

Die Kantenschärfe entspricht dem Gradient einer Kante im Bild. Dieser Gradient sollte dabei nicht vom Minimum bis zum Maximum der Kante gemessen werden, da Kanten unterschiedlich stark abgerundet sind (siehe Abbildung 3.3). Eine Messung vom 20%-Minimum bis zum 80%-Maximum liefert bessere Ergebnisse, da die Anstiegsverläufe in diesem Bereich auch immer annähernd linear sind. Ein solches Anstiegsmaß ist nicht absolut, sondern immer abhängig von den angrenzenden Medien, von der Lage der Kante innerhalb der projizierten Schicht, vom Untersuchungsprotokoll und vom Rekonstruktionskern. Deshalb ist es nur sinnvoll, Gradienten verschiedener Projektionsalgorithmen an genau der gleichen Kante innerhalb der gleichen Schicht des gleichen Datenvolumens miteinander zu vergleichen. Um den Gradienten zu bestimmen, wurde das Profil der Projektion als Mittelwert aller Zeilen, in denen der Quader lag, berechnet. Die Kantenschärfe wurde als der

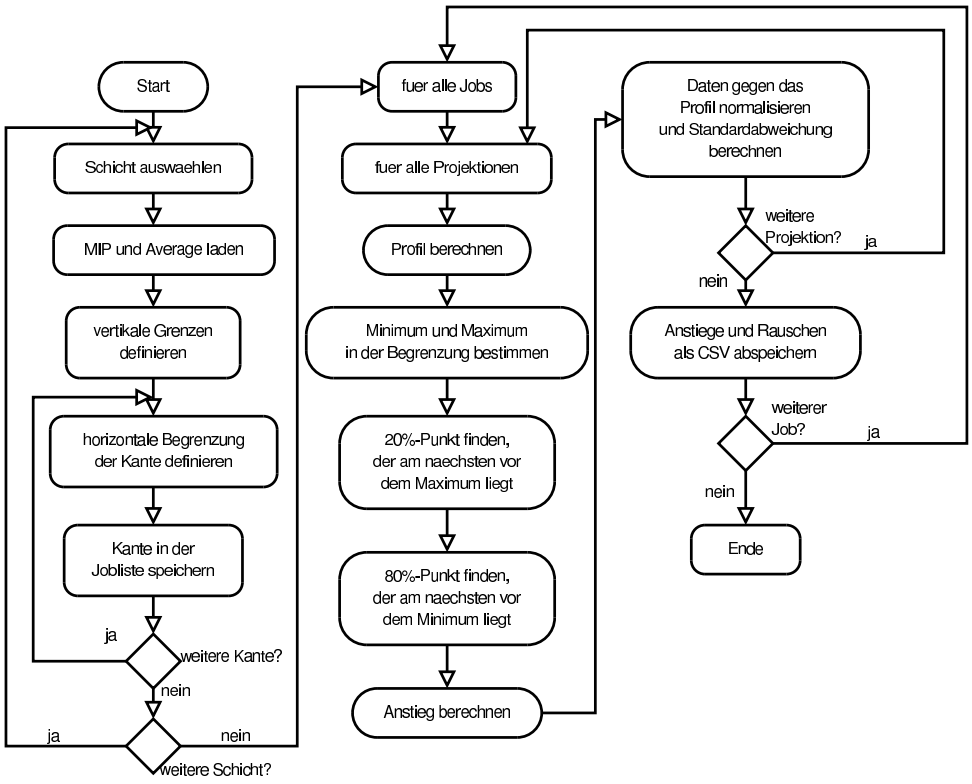
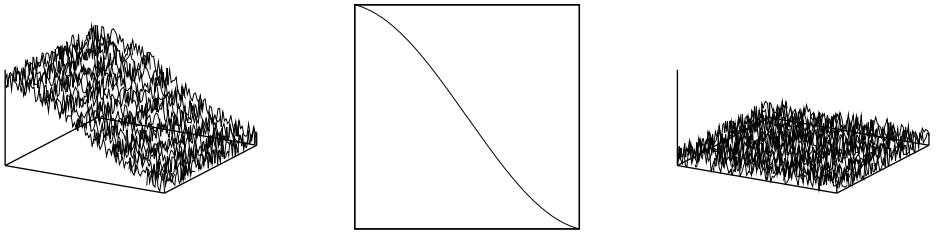


Abbildung 3.4: Ablaufdiagramm von GradientFinder



(a) Rauschen mit überlagerterm Gradienten

(b) Profil

(c) gegen das Profil normalisiertes Rauschen

Abbildung 3.5: Normalisierung des Rauschens

Anstieg vom letzten Punkt des Profils, der 20% über dem Minimum des Übergangs lag, bis zum ersten Punkt des Profils, der 20% unter dem Maximum des Übergangs lag, definiert.

Vor der Berechnung der Standardabweichung wurde von allen Werten des Übergangs das Profil subtrahiert. Dies ermöglichte die Bestimmung des "Rauschens auf der Kante". Abbildung 3.5(a) stellt die ursprünglichen Daten dar, bei denen das Rauschen durch den Gradienten überlagert ist. Durch Mittelung aller Zeilen erhält man das Profil der Daten, welches Abbildung 3.5(b) zeigt. Subtrahiert man dieses von den Daten, so verbleibt nur das Rauschen, dessen Standardabweichung man nun bestimmen kann (Abbildung 3.5(c)).

Die Implementierung des Programmes GradientFinder ist im Anhang B abgedruckt.

3.3.4 MIP-Average-Blending

Um softMip mit einer einfachen Mischung aus Average und MIP zu vergleichen, können Bilderserien generiert werden, in denen eine Average-Projektion in eine MIP der gleichen Schicht übergeblendet wird. Dazu wird einerseits über ein Average-Bild mit ansteigender Intensität ein MIP-Bild der gleichen Schicht gelegt und so ein fließender Übergang von Average zu MIP erzeugt (Blending).

Auch softMip ermöglicht einen solchen fließenden Übergang, indem man

die Wichtungsfunktion $f_w(x)$ zwischen den in 3.1 angegebenen Funktionen $f_w^{\mathrm{MIP}}(x)$ und $f_w^{\mathrm{Average}}(x)$ überblendet.

In beiden Serien lassen sich für jede Kante, wie oben angegeben, die Kantenschärfe und das Bildrauschen bestimmen. Setzt man die Kantenschärfe und das Rauschen von Average auf 0 und die von MIP jeweils auf 1 und normalisiert die gemessenen Werte dementsprechend, so kann man die Werte in ein Rauschen/Kantenschärfe-Diagramm eintragen. Die Fläche unter der Kurve in diesem Diagramm kann als Maß dafür interpretiert werden, wie gut in der jeweiligen Blendingserie Rauschen unterdrückt und Kanten angehoben werden.

3.4 Klinische Studie

3.4.1 Aquisition der Patienten und der Bilddaten

Im Rahmen der urologischen Diagnostik wurden 31 Patienten mit klinischen Zeichen einer Nierenkolik überwiesen. Das Patientenkollektiv setzte sich aus 12 Frauen und 19 Männern im Alter von 23 bis 75 Jahren (Mittelwert: 49 Jahre) zusammen. Die Untersuchungen wurden an einem 16-Zeilen-Spiral-CT[11] in Niedrigdosis-Technik (1 mm Schichtdicke, 3 mAs effektiv, 120 kV, Pitch 1,43, 16 mm Kollimation) vorgenommen. Die resultierende Strahlendosis wurde mit CT-Expo 1.4[12] berechnet. Sie beträgt 0,6 mSv und ist mit der einer konventionellen Röntgenaufnahme vergleichbar[30, 48].

Im Anschluss an die CT-Untersuchung wurden die Bilddaten an eine DicomProjector-Workstation gesendet, welche ins Institutsnetz eingebunden wurde. Dort wurden aus den axialen Dünnschichten frontale Schichten generiert. Dabei wurden Serien mit den Schichtdicken 5 mm, 10 mm und 50 mm in den Projektionen MIP, AVG und softMip – also neun Serien – generiert. Diese Serien wurden im Anschluss automatisch an die Befundungsworkstation übermittelt.

[11] Toshiba Aquillion 16, Toshiba Medical Systems, Nasu, Japan

[12] http://www.mh-hannover.de/fileadmin/kliniken/diagnostische_radiologie/download/ctexpo-d.zip

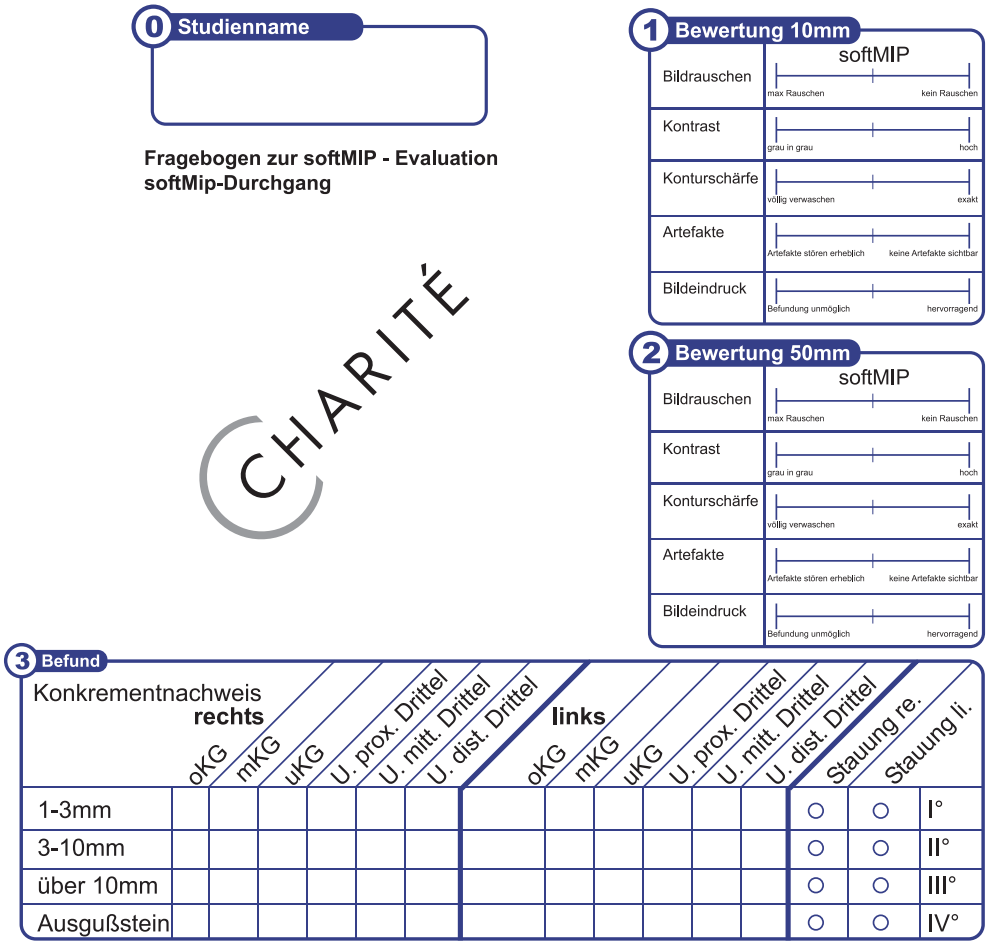


Abbildung 3.6: Befundungsbogen des softMip-Durchgangs

3.4.2 Befundung

Alle Befundungen der klinischen Studie wurden von einem erfahrenen Radiologen durchgeführt. Zur Befundung wurde die Software ViewForum3.1[13] eingesetzt. Die Serien wurden in zwei Durchgängen mittels eines Fragebogens befundet. Im ersten Durchgang wurden die softMip-Serien befundet. Der Fragebogen ist in Abbildung 3.6 dargestellt. Zunächst wurden auf einer Analogskala die subjektiven Bildqualitätsmerkmale Bildrauschen, Kontrast, Konturschärfe, Artefakte und Bildeindruck für die 10 mm- und die 50 mm-

[13]Philips Medical Systems, Eindhoven, Niederlande

Serie erfasst. Darauf folgte eine Befundung bezüglich Anzahl, Größe und Lokalisation evtl. vorhandener Konkremente in den Nieren sowie den Ureteren. Abschließend wurde beurteilt, ob eine Stauung der Nieren vorliegt.

Zeitlich deutlich davon getrennt, wurden die MIP-Average Serien beurteilt. Der Fragebogen dazu ist in Abbildung 3.7 dargestellt. Auch hier wurde zunächst eine subjektive Bewertung durchgeführt. Darauf folgte die Befundung bezüglich Konkrementen und Stauung. Anschließend wurde auch Einsicht in die originalen Dnnschichten der Untersuchung genommen und erneut über Konkremente und Stauung geurteilt. Zum Abschluss wurden dem Untersucher alle Serien vorgelegt, also die originalen Dünnschichten, sowie die MIP-, softMip- und Average-Serien in 10 mm, 50 mm und 100 mm Schichtdicke. Dabei war die Art der Projektion für den Untersucher nicht gekennzeichnet. Die Aufgabe war nun, mit maximal sechs Bildern den erstellten Befund zu dokumentieren. Dabei hatte der Untersucher freie Bildwahl.

Da auch die Befunde der Dünnschichten nicht als Goldstandard, sondern eher als Referenz gelten, wurde in einem "Second-Look" eine korrigierte Referenz als Konsensus aller drei Durchgänge gebildet. Dies war insbesondere nötig, um eine eventuell höhere Sensitivität der softMip detektieren zu können und nicht als schlechte Spezifität fehlzudeuten.

3.4.3 Auswertung

3.4.3.1 Konkremente/StoneSorter

Ausgewertet werden Sensitivität und Spezifität der softMip und der MIP-Average-Kombination gegenüber einzelnen Harnwegskonkrementen. Da sowohl die Größe als auch die Lokalisation der Konkremente in Kategorien eingeordnet werden, besteht die Möglichkeit, dass ein Konkrement, welches sich an der Grenze zweier Kategorien befindet, in einem Durchgang in die eine und im anderen Durchgang in die andere Kategorie eingeordnet wird. Um dieser Tatsache Rechnung zu tragen, wird bei der Auswertung eine Toleranz des softMip sowie des MIP-Average-Durchgangs um eine Kategorie gegenüber der Referenz zugelassen. Wird also beispielsweise im MIP-Average-Durchgang ein Konkrement als 1-3 mm beschrieben und im

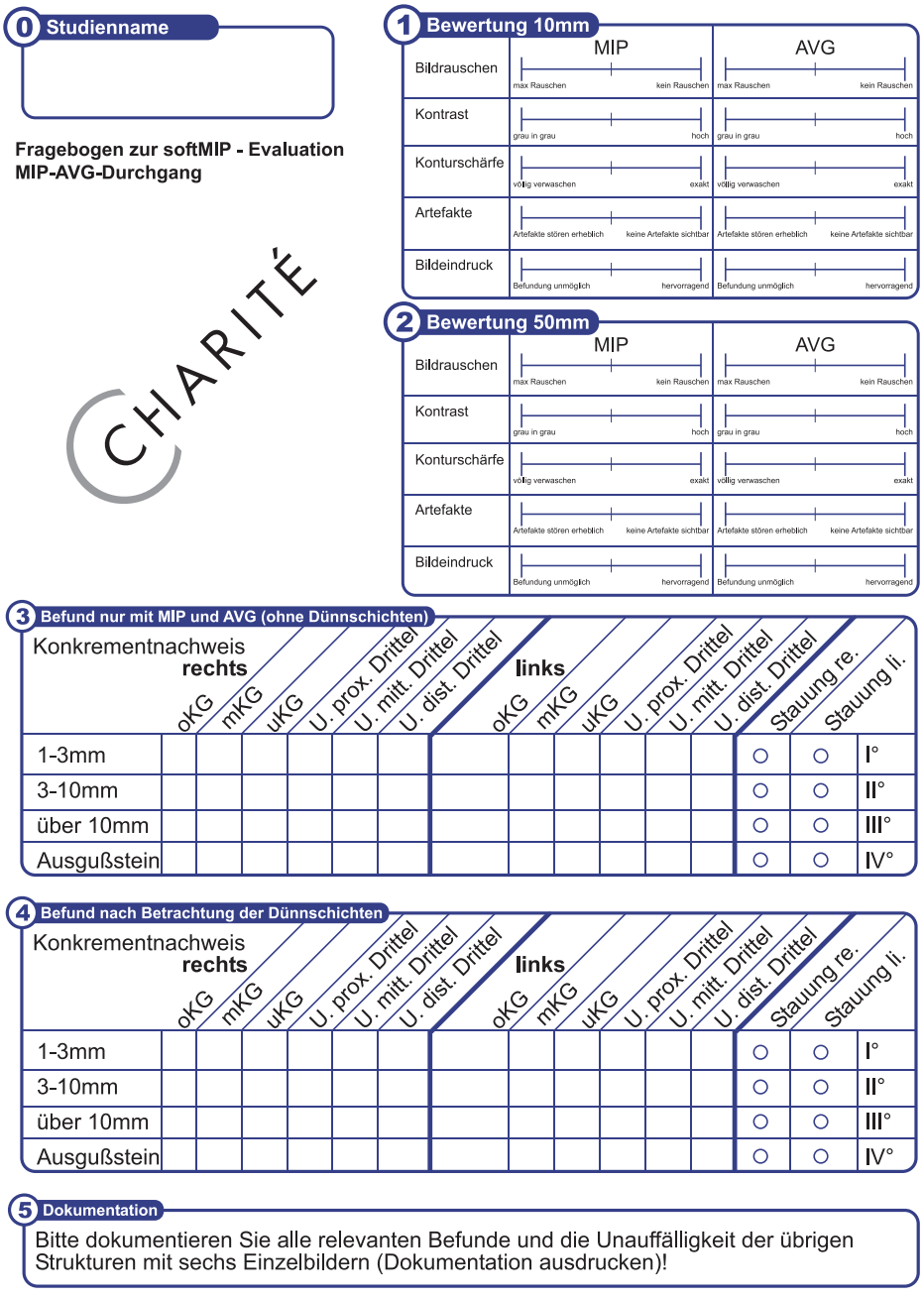


Abbildung 3.7: Befundungsbogen des Mip-Average-Durchgangs

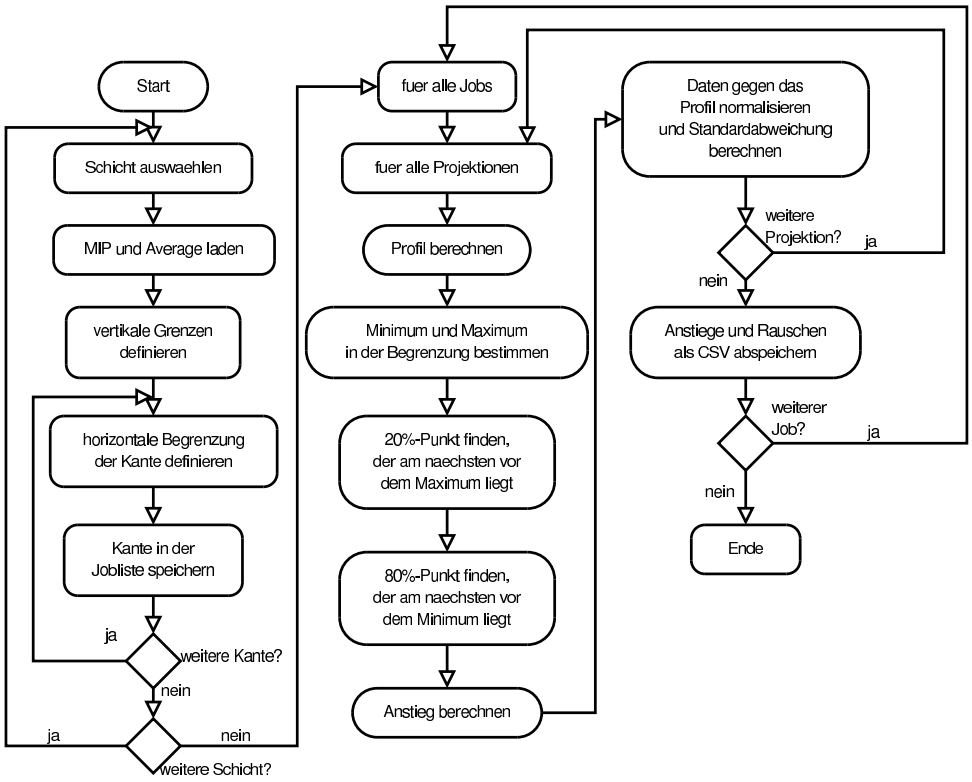


Abbildung 3.8: Ablaufdiagramm von StoneSorter

Referenz-Durchgang als 3-10 mm, so gilt dies dennoch als richtig positiv erkannt.

Diese Auswertung kann bei mehreren Konkrementen pro Niere oder Ureter sehr kompliziert und unübersichtlich werden. Um diese Arbeit möglichst fehlerfrei durchzuführen, wurde das Programm StoneSorter entwickelt. StoneSorter übernimmt diese komplexe Arbeit und ermittelt im festgesetzten Toleranzspielraum die größtmögliche Übereinstimmung zwischen den Untersuchungsgängen und den Referenzdurchgängen. Auch dieses Programm wurde in der Programmiersprache Ruby entwickelt. Abbildung 3.8 zeigt den vereinfachten Programmablauf. Als Eingabe dient eine Auflistung aller Konkremente aller Befundungsdurchgänge aller Patienten. Das Programm gibt am Ende der Berechnung die korrigierte Auflistung aus.

3.4.3.2 Stauung

Bei der Befundung der Bilder wird eine eventuelle Stauung der rechten und linken Niere eingeschätzt. Es werden Sensitivität und Spezifität der softMip und der MIP-Average-Kombination für gestaute Nieren miteinander verglichen. Eine Niere dient dabei als ein Fall, so dass sich 62 Fälle ergeben. Als krank wird eine Niere gewertet, wenn sie eine Stauung ab Grad I aufweist.

3.5 Statistik

Die statistische Auswertung erfolgte mit der Software OpenOffice Calc 1.1.2[14] sowie SPSS 12.0[15]. Statistische Signifikanz wurde bei einer Irrtumswahrscheinlichkeit $p < 0.05$ angenommen.

Unterschiede in Sensitivität und Spezifität in der klinischen Untersuchung wurden mit kombinierten Sensitivitäts- und Spezifitätstabellen nach Hawass auf statistische Signifikanz geprüft [15]. Erst wenn sich in diesem Test ein signifikanter Unterschied ergab, wurden Unterschiede in Sensitivität und Spezifität separat mittels McNemar-Test auf statistische Signifikanz geprüft. Unterschiede in den Gradienten und im Rauschverhalten im Phantomversuch wurden mittels Wilcoxon-Test auf statistische Signifikanz geprüft [1, 32].

In den folgenden Diagrammen sind die Medianwerte dargestellt. Die Fehlerbalken entsprechen den 25%- und 75%-Quartilen.

[14] OpenOffice.org (http://www.openoffice.org)
[15] SPSS Software GmbH, Mnchen (http://www.spss.com/de/)

Kapitel 4

Ergebnisse

4.1 softMip-Server

Als wichtiges Ergebnis – und Grundlage der weiteren Untersuchungen – gelang es, den softMip-Algorithmus als Computerprogramm umzusetzen. Das Programm wurde auf einem Rechner des Instituts eingerichtet. Dieser Rechner wurde im internen Rechnernetz als Dicom-Knoten registriert. Sendet man diesem Dicom-Knoten einen CT-Datensatz, so werden koronale MIP, softMip und Average-Projektionen verschiedener Schichtdicken berechnet und an den Absender des Datensatzes zurückgeschickt. Dieser Vorgang dauert bei einem durchschnittlichen Datensatz mit 500 Dünnschichten ca. drei Minuten.

4.2 Wichtungsfunktionen

In ersten Testversuchen haben schrittweise lineare Funktionen als Wichtungsfunktionen gute Ergebnisse erzielt. Nach einer subjektiven Vorauswahl wurden die in Tabelle 4.1 angegebenen Wichtungsfunktionen ausgewählt, um softMip genauer untersuchen zu können.

Projektion	eingesetzte Funktion f_w
10 mm und 20 mm	$f_w^1(x) = \left\{ \begin{array}{l} 0,5x; \text{ für } x < 0,5 \\ 1,5x - 0,5; \text{ für } x \geq 0,5 \end{array} \right\}$
50 mm und 100 mm	$f_w^5(x) = \left\{ \begin{array}{l} \frac{x}{3}; \text{ für } x < 0,75 \\ 3x - 2; \text{für } x \geq 0,75 \end{array} \right\}$
MIP	$f_w^{\text{MIP}}(x) = \left\{ \begin{array}{l} 1; \text{ für } x > 0,999 \\ 0; \text{ sonst} \end{array} \right\}$
Average	$f_w^{\text{Average}} = 1$

Tabelle 4.1: eingesetzte Wichtungsfunktionen

Gerätename	Anzahl
Aquillion 4	46
Aquillion 16	55
Aquillion 64	47
Tomoscan AV/EU	47
Brilliance 16	45
PQ5000	47
Somatom Sensation 16	45
EBT C-150XP	43
Summe	375

Tabelle 4.2: Anzahl der gemessenen Kanten

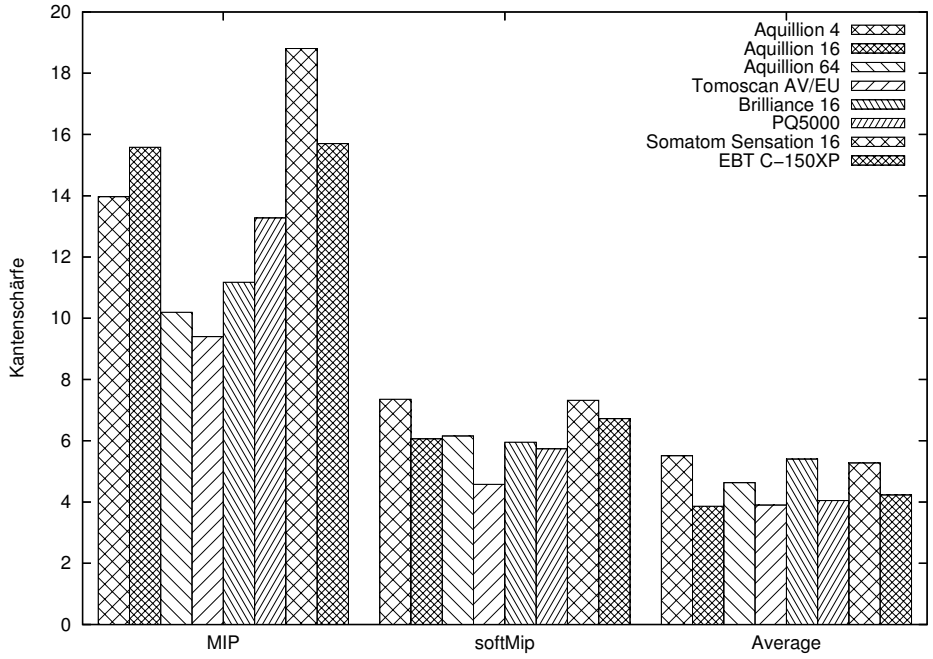


Abbildung 4.1: Kantenschärfe bei den einzelnen Geräten

4.3 Phantomversuch

Je nach Lage des Phantoms im CT-Gerät ergab sich in den Projektionen eine unterschiedliche Anzahl von zu messenden Kanten. Tabelle 4.2 zeigt die Anzahl der gemessenen Kanten für die Serien der einzelnen CT-Geräte. Abbildung 4.1 und Abbildung 4.3(a) zeigen die gemessenen Kantenschärfen, Abbildung 4.2 und Abbildung 4.3(b) die Rauschwerte des Phantomversuches. Es zeigt sich für alle CT-Geräte, dass die Average-Projektion das geringste Rauschen aufweist (7,4). softMip liegt mit dem Rauschen etwas über der Average-Projektion (9,0). Das stärkste Rauschen ist bei der MIP zu beobachten (22,9).

Ähnlich verhält es sich mit der Kantenschärfe. Average hat die geringste Kantenschärfe (4,5), MIP die stärkste (13,2). Die Kantenschärfe der softMip liegt zwischen Average und MIP (6,1). Für alle diese Unterschiede liegt statistische Signifikanz vor ($p < 0,0005$).

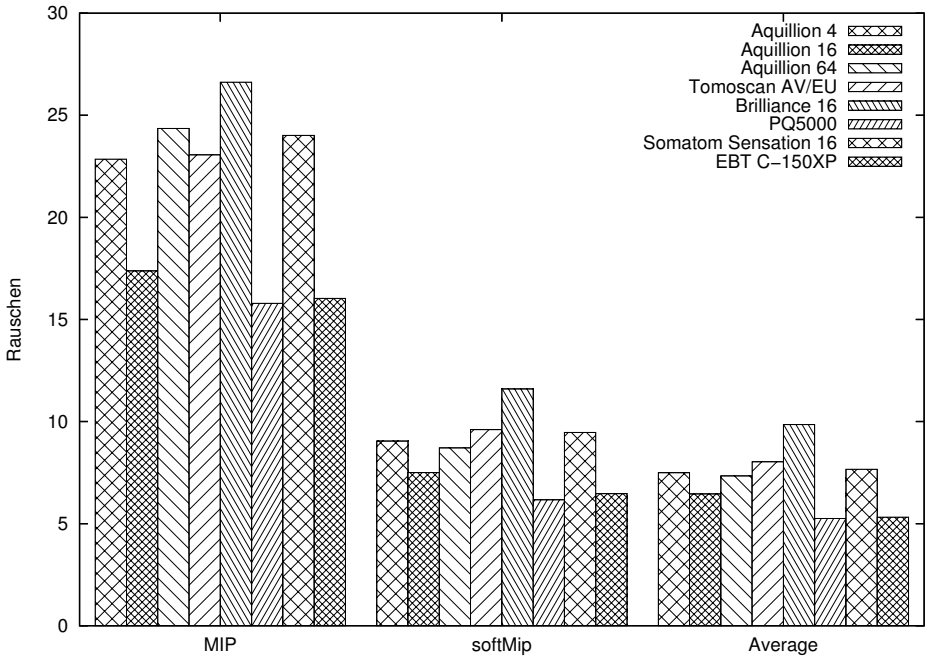
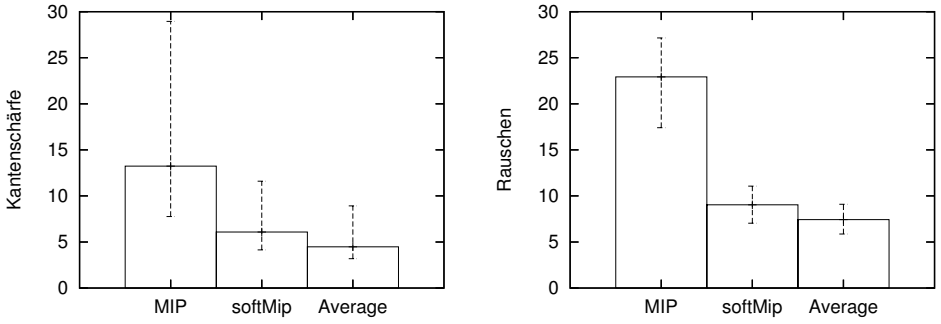


Abbildung 4.2: Rauschen bei den einzelnen Geräten



(a) Gradienten (b) Rauschen

Abbildung 4.3: Kantenschärfe und Rauschen im Phantomversuch

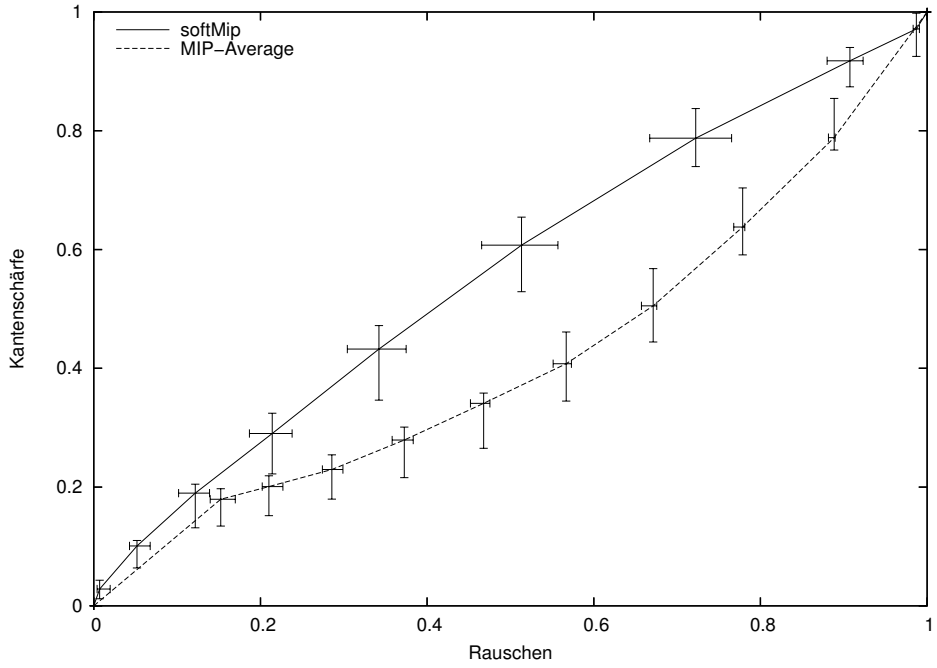


Abbildung 4.4: Blendingkurven von softMip und MIP-Average mit 25% und 75%-Quantil

4.3.1 Blending-Vergleich

Um das softMip-Blending mit dem Mip-Average-Blending zu vergleichen, wurde eine Wichtungsfunktionsschar gewählt, die f_w^{Average} über f_w^1 nach f_w^{MIP} überführt:

$$f_w(x,v) = \left\{ \begin{array}{l} v \cdot f_w^1(x) + (1-v) \cdot f_w^{\text{Average}}(x) \text{ ;für } 0 \leq v < 1 \\ (2-v) \cdot f_w^1(x) + (v-1) \cdot f_w^{\text{MIP}}(x) \text{ ;für } 1 \leq v \leq 2 \end{array} \right\}$$

Diese Funktion bewegt sich von Average ($v = 0$) über softMip ($v = 1$) zur MIP ($v = 2$).

Abbildung 4.4 zeigt das Kantenschärfe-Rauschen-Diagramm für beide Blending-Funktionen im Mittel über alle CT-Geräte. Eine grosse Fläche unter der Kurve in diesem Diagramm zeigt einen Übergang mit geringem Bildrauschen und hoher Kantenschärfe an. Die Fläche unter der Kurve ist in Abbildung 4.5 dargestellt. Für die MIP-Average-Überblendung ergibt sich im Median eine

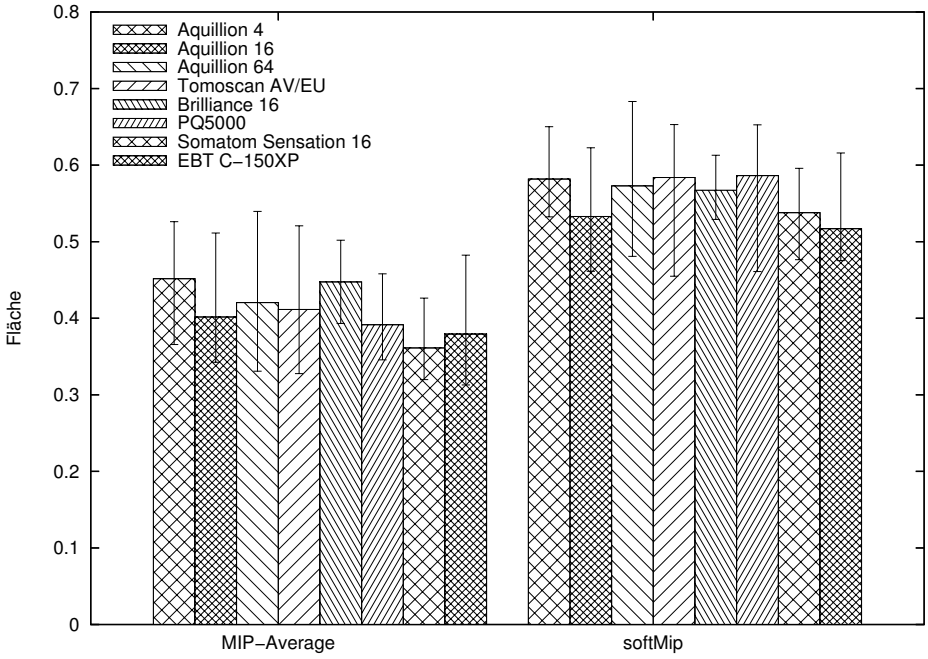
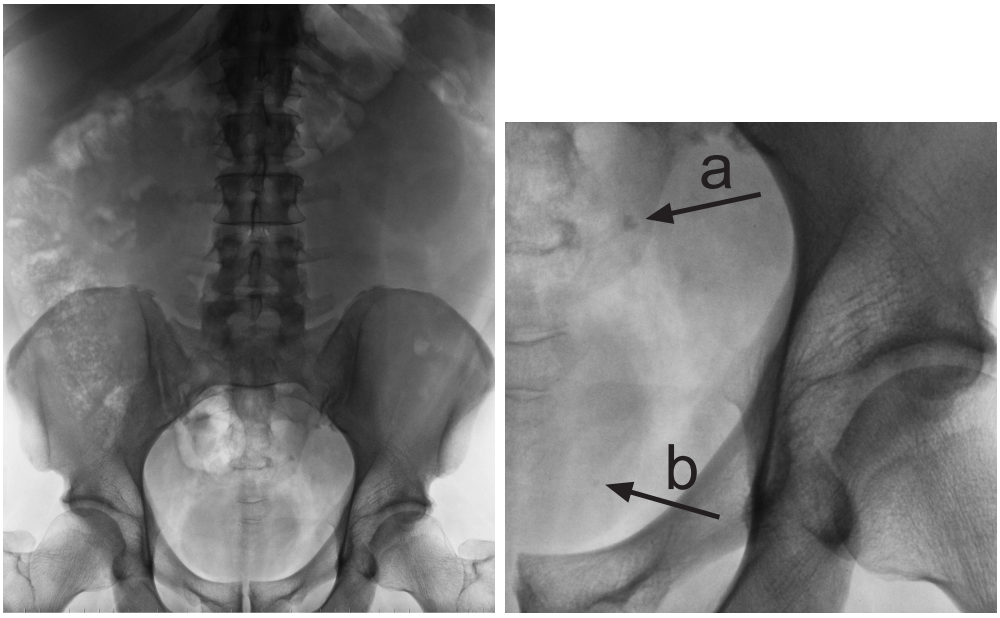


Abbildung 4.5: Fläche unter der Blendingkurve für alle Geräte

Fläche von 0,41. Für die softMip-Überblendung ergibt sich eine Fläche von 0,56. Die Unterschiede wurden für alle Geräte und für die Gesamtheit mit dem Wilcoxon-Test auf statistische Signifikanz geprüft. Dabei ergab sich in jedem Falle $p < 0,0005$.

4.4 Klinische Studie

Abbildung 4.6(a) zeigt die native Nierenübersicht eines 39-jährigen Patienten der klinischen Studie. In Abbildung 4.6(b) wurde das linke kleine Becken herausvergrößert. Die gut sichtbare Struktur a stellt einen Phlebolithen ohne patholgischen Wert dar. Bei der nur schwer abzugrenzenden Struktur b handelt es sich hingegen um ein prävesikales Konkrement, dessen Fund eine wichtige Information für die Kliniker darstellt. Nach Durchführung des Ultra-Low-Dose-CT wurden die in Abbildung 4.7 vergrößert dargestellten koronalen Projektionen angefertigt. Der Phlebolith grenzt sich bei der eingesetzten



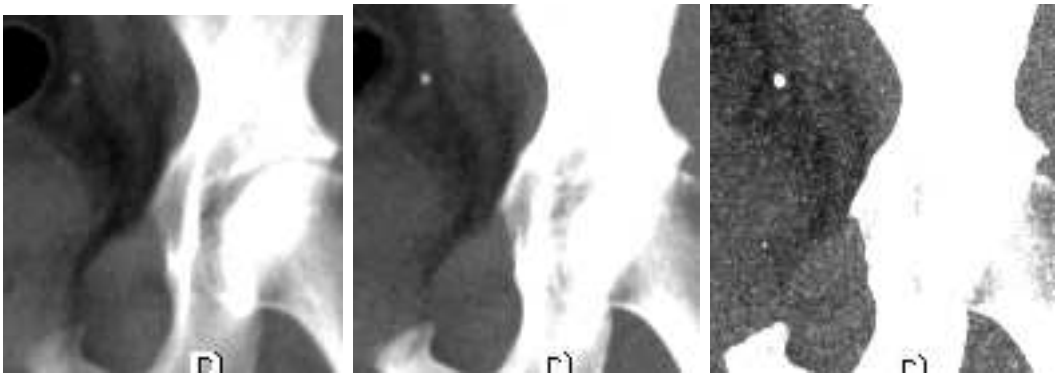
(a) Nierenübersicht (b) Vergrößerung

Abbildung 4.6: prävesikales Konkrement konventionell

Schichtdicke von 50 mm gut in allen Projektionen ab. Jedoch hat man Schwierigkeiten, das Konkrement in der Average-Projektion (Abbildung 4.7(a)) zu finden. In der MIP (Abbildung 4.7(c)) stellt es sich hingegen sehr gut dar. Jedoch ist auch ein starkes Rauschen zu verzeichnen. Die softMip (Abbildung 4.7(b)) hat ein deutlich geringeres Rauschen, das Konkrement ist aber noch immer gut abzugrenzen.

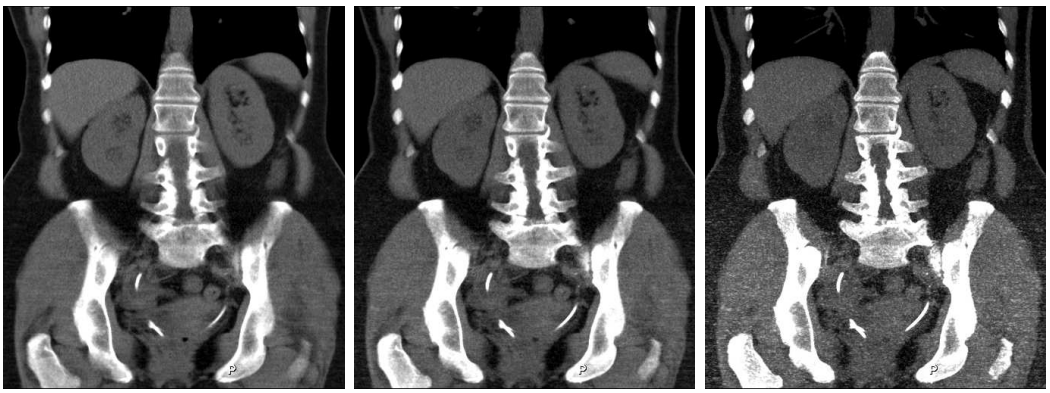
In den Abbildungen 4.8 sind die koronalen Projektionen einer CT-Untersuchung einer 54-jährigen Patientin der Studie dargestellt. Es sind die beidseits liegenden Ureterschienen zu erkennen. Dabei findet sich ein Konkrement im distalen Drittel des rechten Ureters. In der vergrößerten softMip-Projektion in Abbildung 4.9 sieht man in der cranialen Kelchgruppe ein kleines Konkrement. Dieses lässt sich in der Average-Projektion allenfalls erahnen – in der MIP ist es sichtbar, aber nur schwer vom umgebenden Rauschen zu trennen.

Im Rahmen der klinischen Studie ließen sich mit relativ geringem Aufwand softMip-Projektionen beliebiger Datensätze erzeugen und befunden.



(a) Average (b) softMip (c) MIP

Abbildung 4.7: prävesikales Konkrement in den 50 mm CT-Projektionen



(a) Average (b) softMip (c) MIP

Abbildung 4.8: Nierenkonkrement in den 10 mm CT-Projektionen

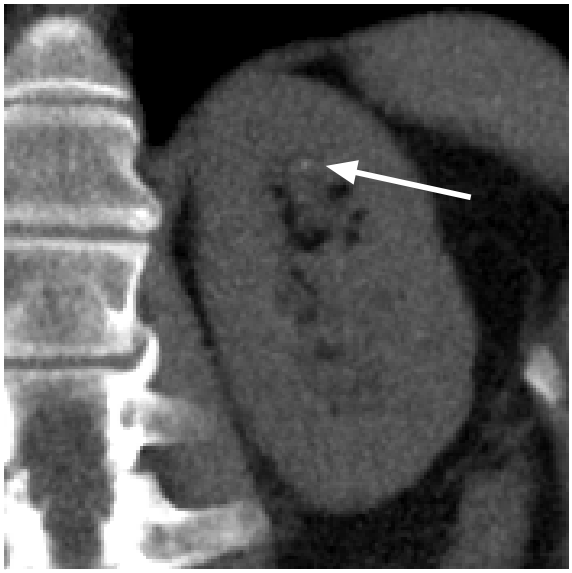


Abbildung 4.9: Nierenkonkrement vergrößert

Dazu wurde der DicomProjektor in das Computernetz des Instituts eingebunden.

4.4.1 Subjektive Bildqualität

Die Werte der verschiedenen Kenndaten der subjektiven Bildqualität für Average, softMip und MIP sind in Abbildung 4.10 für 10 mm Schichten und in Abbildung 4.11 für 50 mm Schichten dargestellt. Die Unterschiede wurden mit dem Wilcoxon-Test auf statistische Signifikanz geprüft. Dabei ergaben sich für 10 mm Schichten die in Tabelle 4.3 und für 50 mm Schichten die in Tabelle 4.4 dargestellten p-Werte.

4.4.2 Konkremente

Insgesamt wurden bei 31 Patienten 62 Nierenkonkremente und 17 Ureterkonkremente gefunden. Im Folgenden ist der Vergleich der diagnostischen Wertigkeit der softMip einerseits und der MIP-Average-Kombination andererseits aufgeführt.

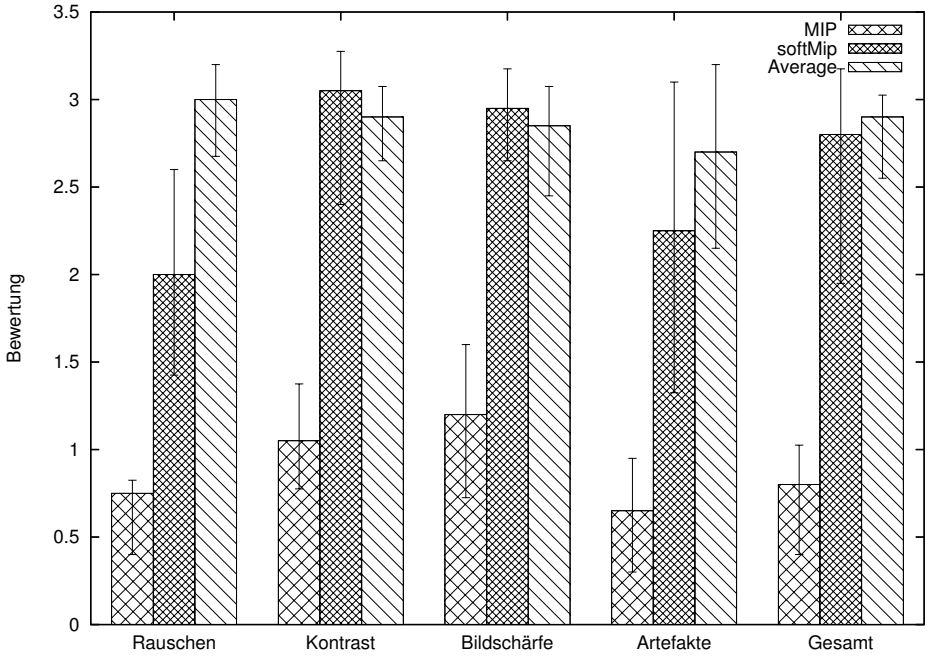


Abbildung 4.10: Bewertung der Bildqualität (Schichtdicke 10 mm)

	MIP vs. AVG	MIP vs. softMip	AVG vs. softMip
Rauschen	$p < 0,0005$	$p < 0,0005$	$p < 0,0005$
Kontrast	$p < 0,0005$	$p < 0,0005$	$p = 0,621$
Bildschärfe	$p < 0,0005$	$p < 0,0005$	$p = 0,537$
Artefakte	$p < 0,0005$	$p < 0,0005$	$p = 0,012$
Gesamt	$p < 0,0005$	$p < 0,0005$	$p = 0,314$

Tabelle 4.3: Unterschiede in der subjektiven Bildqualität (10 mm)

	MIP vs. AVG	MIP vs. softMip	AVG vs. softMip
Rauschen	$p < 0,0005$	$p < 0,0005$	$p < 0,0005$
Kontrast	$p < 0,0005$	$p < 0,0005$	$p = 0,558$
Bildschärfe	$p < 0,0005$	$p < 0,0005$	$p = 0,717$
Artefakte	$p < 0,0005$	$p < 0,0005$	$p = 0,002$
Gesamt	$p < 0,0005$	$p < 0,0005$	$p = 0,754$

Tabelle 4.4: Unterschiede in der subjektiven Bildqualität (50 mm)

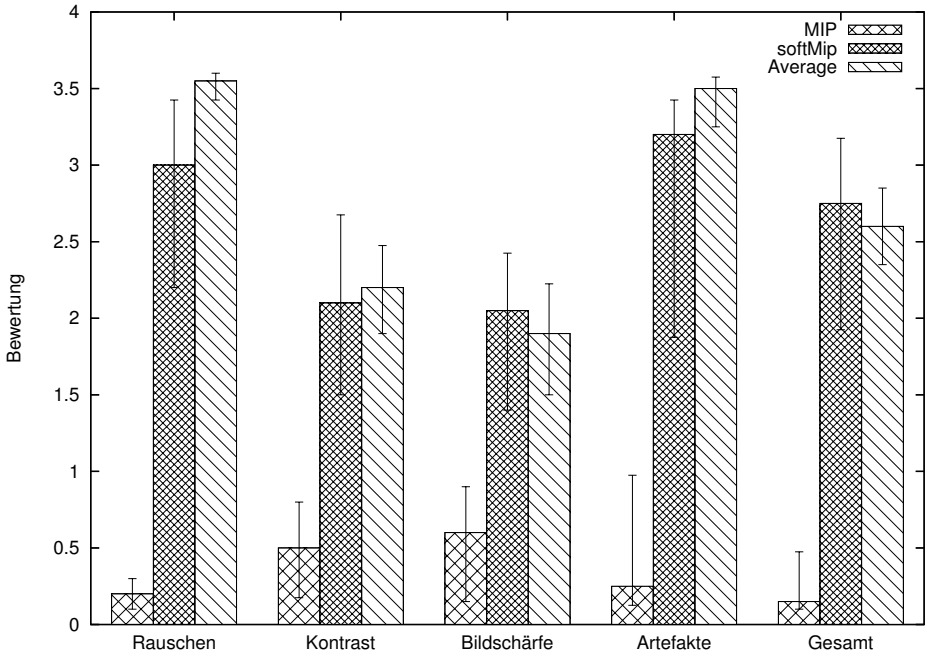


Abbildung 4.11: Bewertung der Bildqualität (Schichtdicke 50 mm)

4.4.2.1 Diagnostische Aussagekraft (Dünnschichten als Referenz)

Für Nierenkonkremente ergab sich die in Tabelle 4.5 dargestellte kombinierte Kreuztabelle für Sensitivität und Spezifität. Dabei stehen RP für "richtig positiv" und FN für "falsch negativ". Der McNemar-Test auf statistische Signifikanz ergibt $p < 0,0005$. Die Kombination aus MIP und Average unterscheidet sich also signifikant in Sensitivität oder Spezifität bezüglich Nierenkonkrementen von der softMip. Die Kreuztabellen zur Sensitivität sind in Tabelle 4.5(a), die zur Spezifität in Tabelle 4.5(b) dargestellt. Es ergeben sich Sensitivitäten von 0,89 für softMip und 0,80 für MIP-Average, $p = 0,219$. Als Spezifitäten ergeben sich für softMip 0,54 und für MIP-Average 1,0 mit $p < 0,0005$.

Die Kreuztabelle für Ureterkonkremente ist in Tabelle 4.7 abgebildet. Der McNemar-Bowker-Test liefert $p = 0,564$. Die Sensitivität lag für softMip bei 0,76, für MIP-Average bei 0,82, doch diese Unterschiede sind nicht statistisch signifikant. Die Spezifitäten wurden für beide Verfahren mit 1,0 bestimmt.

		MIP-Average				
softMip		RP	FP	FN	RN	Gesamt
	RP	36	0	5	0	41
	FP	0	1	0	13	14
	FN	1	0	4	0	5
	RN	0	0	0	14	14
	Gesamt	37	1	9	27	74

Tabelle 4.5: kombinierte Kreuztabelle für Nierenkonkremente mit Dünnschichtreferenz

(a) Sensitivität

		MIP-Average		
softMip		RP	FN	Gesamt
	RP	36	5	41
	FN	1	4	5
	Gesamt	37	9	46

(b) Spezifität

		MIP-Average		
softMip		FP	RN	Gesamt
	FP	0	13	13
	RN	0	15	15
	Gesamt	0	28	28

Tabelle 4.6: Kreuztabellen bezüglich Nierenkonkrementen mit Dünnschichtreferenz

		MIP-Average				
softMip		RP	FP	FN	RN	Gesamt
	RP	12	0	1	0	13
	FP	0	0	0	0	0
	FN	2	0	2	0	4
	RN	0	0	0	19	19
	Gesamt	14	0	3	19	36

Tabelle 4.7: kombinierte Kreuztabelle für Ureterkonkremente mit Dünnschichtreferenz

		MIP-Average				
softMip		RP	FP	FN	RN	Gesamt
	RP	37	0	13	0	50
	FP	0	1	0	4	5
	FN	0	0	12	0	12
	RN	0	0	0	14	14
	Gesamt	37	1	25	18	81

Tabelle 4.8: kombinierte Kreuztabelle für Nierenkonkremente mit Konsensusreferenz

(a) Sensitivität

		MIP-Average		
softMip		RP	FN	Gesamt
	RP	37	13	50
	FN	0	12	12
	Gesamt	37	25	62

(b) Spezifität

		MIP-Average		
softMip		FP	RN	Gesamt
	FP	0	4	4
	RN	0	15	15
	Gesamt	0	19	19

Tabelle 4.9: Kreuztabellen bezüglich Nierenkonkrementen mit Konsensusreferenz

4.4.2.2 Diagnostische Aussagekraft (Konsensus als Referenz)

Im Folgenden sind die Ergebnisse für Sensitivität und Spezifität angegeben, welche sich durch die Anwendung der korrigierten Referenz ergeben haben.

Die kombinierte Kreuztabelle für Nierenkonkremente mit der Konsensusreferenz ist in Tabelle 4.8 dargestellt. Der McNemar-Bowker-Test liefert wieder $p < 0,0005$. Dies bestätigt, dass softMip sich signifikant in Sensitivität oder Spezifität bezüglich Nierenkonkrementen von der Kombination aus MIP und Average unterscheidet. Tabelle 4.8(a) zeigt die Kreuztabelle der Sensitivität, Tabelle 4.8(b) die der Spezifität bezüglich Nierenkonkrementen bei Konsensusreferenz. Daraus ergeben sich die Sensitivitäten von 0,81 für softMip und 0,60 für MIP-Average mit $p < 0,0005$ und die Spezifitäten von 0,79 für softMip und 1,0 für MIP-Average mit $p = 0,125$. Für Ureterkonkremente haben sich beim Einsatz der Konsensusreferenz in Tabelle 4.10 keine neuen Werte ergeben.

		MIP-Average				
softMip		RP	FP	FN	RN	Gesamt
	RP	12	0	1	0	13
	FP	0	0	0	0	0
	FN	2	0	2	0	4
	RN	0	0	0	19	19
	Gesamt	14	0	3	19	36

Tabelle 4.10: kombinierte Kreuztabelle für Ureterkonkremente mit Konsensusreferenz

		MIP-Average				
softMip		RP	FP	FN	RN	Gesamt
	RP	5	0	0	0	5
	FP	0	0	0	3	3
	FN	3	0	1	0	4
	RN	0	0	0	50	50
	Gesamt	8	0	1	53	62

Tabelle 4.11: kombinierte Kreuztabelle für Nierenstauung mit Dünnschichtreferenz

Diese Untersuchung zeigt also bei Nierenkonkrementen eine statistisch signifikant bessere Sensitivität der softMip gegenüber der MIP-Average-Kombination, die Spezifität ist bei softMip tendenziell schlechter als bei MIP-Average – eine statistische Signifikanz konnte hier nicht erzielt werden. Für die diagnostische Aussagekraft bezüglich Ureterkonkrementen konnten keine wesentlichen Unterschiede festgestellt werden.

4.4.3 Stauung

Von 62 Nieren wurden in der Dünnschichtreferenz neun Nieren und in der Konsensusreferenz elf Nieren als gestaut eingestuft.

4.4.3.1 Diagnostische Aussagekraft (Dünnschichten als Referenz)

Tabelle 4.11 zeigt die kombinierte Kreuztabelle für die Sensitivität und Spe-

(a) Sensitivität

softMip \ MIP-Average	RP	FN	Gesamt
RP	5	0	5
FN	3	1	4
Gesamt	8	1	9

(b) Spezifität

softMip \ MIP-Average	FP	RN	Gesamt
FP	0	3	3
RN	0	50	50
Gesamt	0	53	53

Tabelle 4.12: Kreuztabellen bezüglich Nierenstauung mit Dünnschichtreferenz

zifität mit den Dünnschichten als Referenz. Es ergibt sich für softMip eine Sensitivität von 0,56. Für die MIP-Average-Kombination ergibt sich die Sensitivität 0,89. Die Spezifität liegt für softMip bei 0,94, die für MIP-Average bei 1,00. Der McNemar-Test ergab $p = 0,05$. Somit konnte für die aufgetretenen Unterschiede keine statistische Signifikanz nachgewiesen werden. Die genauere Analyse mittels getrennter Kreuztabellen für Sensitivität (Tabelle 4.11(a)) und Spezifität (Tabelle 4.11(b)) bestätigt dies. Der McNemar-Test liefert $p = 0,25$ für die Unterschiede in der Sensitivität und $p = 0,25$ für die Unterschiede der Spezifität.

4.4.3.2 Diagnostische Aussagekraft (Konsensus als Referenz)

Die Benutzung der korrigierten Referenz zur Berechnung der Sensitivität und Spezifität lieferte nur gering abweichende Werte. Der Vergleich der Ergebnisse von softMip und MIP-Average führte zu der kombinierten Kreuztabelle 4.13. Die Unterschiede zur Dünnschichtreferenz sind gering. Es ergaben sich die Sensitivitäten von 0,64 und 0,73 und die Spezifitäten von 0,98 und 1,00 für softMip und MIP-Average. Der McNemar-Test ergab hier $p = 0,549$. Die Unterschiede in diesen Kenndaten sind nicht statistisch signifikant.

4.4.4 Dokumentation

Abbildung 4.12 zeigt die relativen Häufigkeiten der einzelnen Projektionen in der Dokumentation. MIP-Bilder wurden nie zur Dokumentation verwendet. Mittels Wilcoxon-Test wurde geprüft, ob es Unterschiede in der Häufigkeit

		MIP-Average				
softMip		RP	FP	FN	RN	Gesamt
	RP	5	0	2	0	7
	FP	0	0	0	1	1
	FN	3	0	1	0	4
	RN	0	0	0	50	50
	Gesamt	8	0	3	51	62

Tabelle 4.13: kombinierte Kreuztabelle für Nierenstauung mit Konsensusreferenz

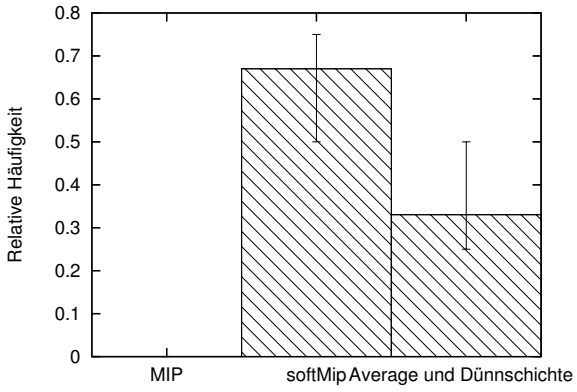


Abbildung 4.12: relative Häufigkeiten der Projektionen in der Dokumentation

der softMip-Projektionen gegenüber der der restlichen Projektionen zusammen gab. Es ergab sich eine relative Häufigkeit von 0,67 für softMip und von 0,33 für Average und Dünnschichten zusammen, $p = 0,009$.

Kapitel 5

Diskussion

5.1 Computertomographie und Strahlenexposition

Die Computertomographie ist ein wesentlicher Bestandteil moderner klinischer Bildgebung. Die technischen Entwicklungen des letzten Jahrzehnts bis hin zur MSCT haben die Möglichkeiten dieser Modalität enorm gesteigert. Die CT hat weiterhin nur einen relativ geringen Anteil an radiologischen Untersuchungen, ist aber gleichwohl für ein Drittel der diagnostisch bedingten Strahlenexposition verantwortlich [36]. Da die Strahlenexposition und der Strahlenschutz immer mehr in den Vordergrund treten, wurden wesentliche Anstrengungen unternommen, die Strahlenexposition der CT zu reduzieren.

Insbesondere für Untersuchungen an Kindern wurden schon früh Untersuchungsprotokolle mit verminderter Strahlenexposition (Low-Dose-CT) entwickelt und untersucht [17, 49, 53]. Low-Dose-CT-Untersuchungen des Kopfes, der Lunge und des Beckenskeletts benötigen nur 50% der Dosis herkömmlicher Untersuchungen, ohne die Beurteilbarkeit der Bilder zu beeinträchtigen [16, 35, 43, 55]. Cohen et al. und Iannaccone et al. bewerteten die Möglichkeiten der Low-Dose-CT-Koloskopie [7, 19]. Dabei kamen sie zu dem Ergebnis, dass es schon ab ca. 2 mSv Strahlenexposition sinnvoll möglich ist, diese Untersuchungen durchzuführen.

Kalra et al. kamen zu dem Ergebnis, dass bei Low-Dose-Abdomenuntersuchungen,

insbesondere bei übergewichtigen Patienten, die Bildqualität durch verstärktes Bildrauschen sehr stark leidet. So waren in Einzelfällen die Bilder nicht beurteilbar [26].

Jüngere Studien gehen mit der Ultra-Low-Dose-CT (ULDCT) den nächsten Schritt. Diese Untersuchungen weisen eine Strahlenexposition auf, die mit der einer konventionellen Röntgenaufnahme der untersuchten Region vergleichbar ist. ULD-Thorax-Untersuchungen haben ergeben, dass auch bei einer Dosisreduktion um 93,3% die Detektion pulmonaler Rundherde nicht wesentlich eingeschränkt ist [50]. Untersuchungen zur Detektion von Harnwegskonkrementen zeigen eine diagnostische Überlegenheit der Abdomen-ULDCT gegenüber dem Ultraschall [30, 48]. Durch die Minimierung der Strahlenexposition steigt das Bildrauschen stark an, und erschwert so die Beurteilung.

5.2 softMip-Ergebnisse

Der verfolgte Ansatz dieser Arbeit, einen Projektionsalgorithmus zu entwickeln und mittels eines eigenen Dicom-Projektionsservers zu evaluieren, zeigte sich als ein gangbarer Weg zur Untersuchung neuer computergestützter Nachverarbeitungsverfahren. Ein wesentlicher Vorteil dieses Weges ist die Unabhängigkeit von Rekonstruktions- und Befundungssoftware und den entsprechenden Workstations. Der softMip-Projektionsserver konnte erfolgreich als zusätzliche Funktionalität in das bestehende heterogene Radiologie-Netz integriert werden. Langfristig ist aber eine Integration in bestehende Befundungssoftware wünschenswert.

Die Ergebnisse der Arbeit zeigen, dass durch den Einsatz von softMip in Niedrigdosisdatensätzen das Bildrauschen verringert und die Kontraste erhöht werden können. Dadurch ermöglicht softMip eine bessere Beurteilbarkeit pathologischer Strukturen. Die softMip ist kein festgelegtes Projektionsverfahren, denn die Charakteristik ist sehr stark von der Wichtungsfunktion f_w abhängig. Somit ergibt sich eine große Gruppe von Projektionen, die von der minMip über die Average-Projektion bis zur MIP reicht und alle erdenklichen Zwischenstufen einschließt. Der Blending-Vergleich zeigt, dass für die enstandenen Bilder – aus der für diese Arbeit gewählten f_w(siehe Tabelle 4.1)

– die Kantenschärfe näher an der Kantenschärfe der Average-Projektion liegt als an der hohen Kantenschärfe der MIP. Das Verhältnis von Bildrauschen zu Kantenschärfe ist in den softMip-Bildern jedoch sehr gut, da das Bildrauschen der softMip-Bilder noch näher an dem der Averageprojektion liegt. Das optimale Verhältnis von Rauschen und Kantenschärfe wurde bei dieser Wahl von f_w sicher noch nicht erreicht und muss auch für jede Fragestellung neu ermittelt werden.

Zu der klinischen Studie ist zu bemerken, dass sich die subjektive Bildqualität der softMip kaum von der hohen Qualität der Average-Projektion unterscheidet. Damit erfüllt die softMip die wesentliche Forderung, ein für den Betrachter angenehmes und leicht zu beurteilendes Bild zu erzeugen. Die durchgehend schlechte Beurteilung der Bildqualität der MIP-Bilder zeigt eindrucksvoll, dass die MIP kein geeignetes Projektionsverfahren für stark verrauschte Datensätze der Ultra-Low-Dose-Bildgebung ist.

Dass die softMip es vermag, bei gleichzeitiger Rauschunterdrückung auch die Kontraste anzuheben und somit Konkremente gut sichtbar darzustellen, zeigt die signifikant höhere Sensitivität der softMip gegenüber der MIP-Average-Kombination für Nierenkonkremente. Die Kontrastanhebung in den MIP-Bildern blieb offenbar wirkungslos, da die kontrastangehobenen Konkremente im verstärkten Rauschen untergingen. Allerdings kann dagegen gehalten werden, dass die Spezifität der softMip für Nierenkonkremente tendenziell niedriger war als die der MIP-Average-Kombination. Für diesen Fall wäre eine ROC-Analyse wünschenswert, da man bei vielen diagnostischen Tests durch Verschiebung des Cutoff-Punktes auf der ROC-Kurve die Sensitivität erhöhen und die Spezifität senken kann und umgekehrt [6, 10]. Eine solche Auswertung ist aber nur bei kontinuierlichen Messwerten sinnvoll möglich und gestaltet sich bei binären Detektionsentscheidungen für Konkremente sehr aufwendig. Andererseits könnte die verringerte Spezifität auch darauf zurückzuführen sein, dass die Untersucher mit der softMip tatsächlich Konkremente fanden, die selbst in den Dünnschichten nicht zu detektieren waren. Studien mit höheren Fallzahlen und eventuell auch die Erfassung von ROC-Kurven sind hier nötig. Für Ureterkonkremente sind die Ergebnisse nur schwer zu werten, da in allen Patienten zusammen nur 17 Konkremen-

te gefunden wurden. Die Ergebnisse zeigen keine wesentlichen Unterschiede in Sensitivität und Spezifität bezüglich Ureterkonkrementen. Interessant ist, dass in beiden Gruppen die Spezifität bei 1,0 liegt – es also keine falsch positiven Ureterkonkremente gab.

Ähnlich verhält es sich bei der Frage nach einer Nierenstauung. Bei nur elf Patienten wurde eine Stauung diagnostiziert – es wurden keine verwertbaren Unterschiede der diagnostischen Wertigkeit von softMip und MIP-Average gefunden.

Eine große Stärke der softMip-Bilder zeigte sich bei der Dokumentation der Befunde und deren Lokalisation. Die Bilder der softMip-Serien wurden überdurchschnittlich häufig zur Dokumentation der Befunde eingesetzt. Diese Tatsache legt nahe, dass softMip-Bilder pathologische Befunde anschaulicher zeigen als MIP- oder Average-Bilder. Außerdem ermöglichen die softMip-Bilder eine sehr gute anatomische Einordnung der Befunde.

5.3 Vergleich mit ähnlichen Projekten

5.3.1 Rauschminderung in den axialen Schichten

Kalra et al. berichten über die Anwendung von Rauschminderungsfiltern in Low-Dose-CT Untersuchungen [27]. Als Ergebnis zeigt sich, dass die Filter die subjektive Bildqualität durchaus steigern. Im Gegensatz zu den Ergebnissen der softMip-Untersuchung sind pathologische Strukturen mitunter schlechter auszumachen. Auch ändert sich die gesamte Bildcharakteristik durch Änderungen der Modulationstransferfunktion (MTF) im hohen Frequenzbereich. Wedegartner et al. untersuchten den Einfluß der Rekonstruktionsschichtdicke auf die Bildqualität und die diagnostische Wertigkeit der Bilder [56]. Es zeigte sich, dass für Leberaufnahmen dünne Schichten (3 mm) die genaueste diagnostische Information liefern, auch wenn in dickeren Schichten das Rauschen besser reduziert wird. Keselbrener et al. beschreiben die Anwendung nichtlinearer Filter, denen zwar komplexe Berechnungen zugrunde liegen. Die resultierenden Bilder zeigen aber zumindest bei Phantomen sehr gute Ergebnisse [29].

5.3.2 Rauschminderung vor der Rekonstruktion

Neben der Rauschminderung in den rekonstruierten Bilddaten ist es auch möglich, das Rauschen bereits zuvor, im Rekonstruktionsverfahren oder im Akquisitionsprozess, zu reduzieren [3, 14, 21, 52]. Dieses sehr komplexe Thema würde aber den Rahmen dieser Arbeit sprengen, da diese sich ausschließlich mit der Nachverarbeitung bereits rekonstruierter Bilder befasst.

5.3.3 Auflösung und Bildrauschen

Viele Verfahren zu Verminderung des Bildrauschens beruhen letztendlich auf der Erhöhung der Sampling-Frequenz – der Anzahl der Messpunkte, die einen Bildpunkt ergeben. Dadurch wird aber gleichzeitig auch die räumliche Auflösung der Bilder gemindert. Dies gilt prinzipiell auch für Projektionsverfahren wie Average und softMip, welche auch das Rauschen mindern. Die Auflösung wird bei diesen Verfahren jedoch nur in der Achse gemindert, die senkrecht zum Bild steht. Die Auflösung in X-Y-Richtung ist so hoch wie in den Ursprungsdaten. Es ist also auch in solchen Bildern die Ausdehnung oder Lokalisation bestimmter Strukturen in die Bildebene hinein nur schwer zu bestimmen. Um diese Einschränkung zu umgehen, kann man aber Average- oder auch softMip-Projektionen, wie bei der MPR, in mehreren Schnittführungen berechnen und erhält im Resultat Bilder mit gemindertem Rauschen und voller Auflösung in der neuen Schnittebene. Sieht man sich zunächst koronale Schichten an, und ist sich unsicher über die Ausdehnung eines Befundes in der Blickrichtung, so kann man diese Frage sehr leicht auf sagittalen oder axialen Bildern beantworten. Mit einer solchen Vorgehensweise kann die eingeschränkte Auflösung projektionsbasierter Bilder umgangen werden.

5.3.4 Anwendungen der MIP

Die MIP wurde ursprünglich für die Beurteilung von MR-Angiographien entwickelt [28, 33]. Auch für die Suche nach Lungenrundherden und die Beurteilung des Bronchialsystems hat sich die MIP als hilfreich erwiesen [39]. Eine wichtige Domäne der MIP ist aber die CT-Angiographie, da durch die selek-

tive Darstellung der dichten Voxel eine gute Darstellung der kontrastierten Gefäße erreicht wird [37]. In bestimmten Situationen ist es möglich, dass auf MIP-Bildern intraluminale Thromben in seitlich getroffenen Gefäßen nicht sichtbar sind, obwohl diese in den axialen Dünnschichten abgebildet sind [34, 46]. Da die softMip im Gegensatz zur MIP auch eine Mittelwertbildung der Schichten einschließt, ist es möglich, dass intraluminale Thromben auf softMip-Bildern nicht maskiert werden.

5.4 Limitierungen

5.4.1 Clusterfehler

Da im Laufe der Untersuchung bei einigen Patienten mehrere Konkremente diagnostiziert wurden und diese Konkremente in der Auswertung als einzelne Fälle gewertet werden, vervielfachen sich auch die ausschließlich mit diesem Patienten verbundenen Auswertungsfehler. So tritt beispielsweise bei adipösen Patienten verstärkt Rauschen auf, was zu einer schlechteren Konkrementerkennung führt. Dies kann Fehler in verschiedener Form verursachen. Einerseits werden Konkremente leichter übersehen, da sie im Rauschen nicht zu erkennen sind. Andererseits kann es vorkommen, dass besonders starke Rauschpunkte als Konkrement fehlgedeutet werden. So kann ein Fehler – z.B. das zu starke Bildrauschen bei einem Patienten – gleich mehrere Fälle verfälschen.

5.4.2 Toleranz bei Konkrementgröße und -lokalisation

In der vorliegenden Studie war vorgesehen, jedes Konkrement als einzelnen Fall zu werten. Um mehrere Konkremente pro Patient in den verschiedenen Befundungsdurchgängen möglichst genau zuordnen zu können, mussten die Konkremente charakterisiert werden. Dazu wurden für jedes Konkrement die Größe und die Lokalisation kategorisiert. Anhand dieser Einordnung war es bei der Auswertung möglich, die Konkremente einander zuzuordnen.

Die Bestimmung der Lokalisation eines Nieren- oder Ureterkonkremen-

tes ist in den axialen Dünnschichten deutlich ungenauer als in den koronalen Projektionen, da die kranio-kaudale Ausdehnung in axialen Schichten nur abgeschätzt werden kann – in koronalen Schichten kann man sie hingegen direkt erkennen. Während der Auswertung wurde deshalb bei der Konkrementlokalisation eine Abweichung von einem Segment toleriert. Dies ist wichtig, um die unterschiedlich gute Lokalisierbarkeit in den frontalen Projektionen und den axialen Dünnschichten auszugleichen. Gleichzeitig aber erhöht sich dadurch auch das Risiko, Konkremente als identisch zu bewerten, welche evtl. nicht identisch sind. In den unterschiedlichen Projektionen wird die Größe desselben Konkrementes unterschiedlich eingeschätzt. Auf MIP-Bildern wird sie eher überschätzt, auf Average-Bildern werden Konkremente eher zu klein sichtbar. So wurde in der vorliegenden Untersuchung eine Größenabweichung um eine Größenkategorie toleriert. Konkremente, die in Größe oder Lokalisation im Grenzbereich zweier Kategorien befindlich sind und daher bei mehreren Durchgängen in verschiedene Kategorien eingeordnet werden, können durch diese Toleranzen dennoch als identisch erkannt werden.

5.4.3 Korrigierte Referenz

Da für die Detektion von Konkrementen kein Goldstandard verfügbar ist, wurden die Befunde der Dünnschicht-Serien als Referenz benutzt. In einem weiteren Durchgang wurde eine korrigierte Referenz als Konsensus aller Serien geschaffen. Eventuelle Fehler, wie ein falsch positives Konkrement in den softMip-Serien, könnten bei der Konsensusbildung auch in die korrigierte Referenz eingehen. Dies hätte zur Folge, dass das falsch positive Konkrement als richtig positiv bewertet würde.

5.4.4 Tools und Scripte

Die Auswertung der Untersuchung erfolgte mithilfe zweier Ruby-Scripte[1]. Diese Scripte wurden explizit für diese Arbeit entwickelt. Dadurch sind sie als neue Auswertungsmethoden anzusehen, die noch nicht etabliert und we-

[1] GradientFinder, StoneSorter

nig getestet sind. Die Auswertung wird durch die Verwendung dieser Scripte selbst für medizinisch versiertes Personal schwer nachvollziehbar. Andererseits ist eine computergestützte Auswertung sehr objektiv und damit sehr reliabel.

5.5 Ausblick

Die weitere sinnvolle Entwicklung der softMip erfasst die systematische Untersuchung verschiedener Wichtungsfunktionen f_w in verschiedenen Anwendungsbereichen. Es ist auch vorstellbar, die Wichtungsfunktion mittels Phantomen computergestützt automatisch zu optimieren, so dass ein einstellbarer Kompromiss aus Kantenschärfe und Bildrauschen resultiert.

Wünschenswert wären auch umfassendere Studien zur Untersuchung der Bildqualität und der diagnostischen Wertigkeit der softMip, mit einerseits mehreren befundenden Radiologen und andererseits noch mehr Fällen. Andere sinnvolle Anwendungsgebiete für die softMip sollten untersucht werden. Die softMip könnte evtl. sinnvoll bei CT der Nasennebenhöhlen, des Thorax und des Skeletts eingesetzt werden. Auch für CT-Angiographien kann die softMip gute Dienste leisten, ist doch hier die MIP ein verbreitetes Nachverarbeitungsverfahren. Mit der softMip könnte die Bildqualtät gesteigert werden. Gleichzeitig könnte der Mangel der MIP, in bestimmten Situationen Pathologien durch davor oder dahinter befindliche dichte Strukturen zu verbergen, durch die bei der softMip vorhandene Mittelwertbildung ausgeglichen werden.

Literaturverzeichnis

[1] Applegate, K. E. ; Tello, R. ; Ying, J. : Hypothesis testing III: counts and medians. In: *Radiology* 228 (2003), Nr. 3, S. 603–8

[2] Asplin, J. R. ; Coe, F. L. ; Favus, M. J.: *Harrison's Principles of Internal Medicine*. Bd. 2. The McGraw-Hill Companies, Inc., 2003

[3] Badea, C. ; Gordon, R. : Experiments with the nonlinear and chaotic behaviour of the multiplicative algebraic reconstruction technique (MART) algorithm for computed tomography. In: *Phys Med Biol* 49 (2004), Nr. 8, S. 1455–74

[4] Becker, C. R. ; Schatzl, M. ; Schoepf, U. J. ; Bruning, R. ; Reiser, M. F.: Technische Grundlagen und Akquisitionsbedingungen der Elektronenstrahl-Computertomographie. In: *Radiologe* 38 (1998), Nr. 12, S. 987–92

[5] Brix, G. ; Nagel, H. D. ; Stamm, G. ; Veit, R. ; Lechel, U. ; Griebel, J. ; Galanski, M. : Radiation exposure in multi-slice versus single-slice spiral CT: results of a nationwide survey. In: *Eur Radiol* 13 (2003), Nr. 8, S. 1979–91

[6] Brown, M. D. ; Reeves, M. J.: Evidence-based emergency medicine/skills for evidence-based emergency care. Interval likelihood ratios: another advantage for the evidence-based diagnostician. In: *Ann Emerg Med* 42 (2003), Nr. 2, S. 292–7

[7] Cohnen, M. ; Vogt, C. ; Beck, A. ; Andersen, K. ; Heinen, W. ; vom Dahl, S. ; Aurich, V. ; Haeussinger, D. ; Moedder, U. :

Feasibility of MDCT Colonography in ultra-low-dose technique in the detection of colorectal lesions: comparison with high-resolution video colonoscopy. In: *AJR Am J Roentgenol* 183 (2004), Nr. 5, S. 1355–9

[8] Diederich, S. ; Wormanns, D. : Impact of low-dose CT on lung cancer screening. In: *Lung Cancer* 45 Suppl 2 (2004), S. S13–9

[9] Eibel, R. ; Bruning, R. ; Schopf, U. J. ; Leimeister, P. ; Stadie, A. ; Reiser, M. F.: Bildanalyse bei der Mehrschicht-Spiral-CT der Lunge mit MPR- und MIP-Rekonstruktionen. In: *Radiologe* 39 (1999), Nr. 11, S. 952–7

[10] Farr, B. M. ; Shapiro, D. E.: Diagnostic tests: distinguishing good tests from bad and even ugly ones. In: *Infect Control Hosp Epidemiol* 21 (2000), Nr. 4, S. 278–84

[11] Galanski, M. ; Nagel, H. D. ; Stamm, G. : CT-Expositionspraxis in der Bundesrepublik Deutschland. In: *Fortschr Röntgenstr* 173 (2001), Nr. 10, S. R1–66

[12] Gies, M. ; Kalender, W. A. ; Wolf, H. ; Suess, C. : Dose reduction in CT by anatomically adapted tube current modulation. I. Simulation studies. In: *Med Phys* 26 (1999), Nr. 11, S. 2235–47

[13] Greess, H. ; Wolf, H. ; Suess, C. ; Kalender, W. A. ; Bautz, W. ; Baum, U. : Dosisautomatik bei der Mehrzeilenspiral-CT: Phantommessungen und klinische Ergebnisse. In: *Fortschr Röntgenstr* 176 (2004), Nr. 6, S. 862–9

[14] Harpen, M. D.: A computer simulation of wavelet noise reduction in computed tomography. In: *Med Phys* 26 (1999), Nr. 8, S. 1600–6

[15] Hawass, N. E.: Comparing the sensitivities and specificities of two diagnostic procedures performed on the same group of patients. In: *Br J Radiol* 70 (1997), Nr. 832, S. 360–6

[16] Hein, E. ; Rogalla, P. ; Klingebiel, R. ; Hamm, B. : Low-dose CT of the paranasal sinuses with eye lens protection: effect on image quality and radiation dose. In: *Eur Radiol* 12 (2002), Nr. 7, S. 1693–6

[17] Honnef, D. ; Wildberger, J. E. ; Stargardt, A. ; Hohl, C. ; Barker, M. ; Gunther, R. W. ; Staatz, G. : Mehrschicht-Spiral-CT (MSCT) in der Kinderradiologie: Dosisreduktion bei der Untersuchung von Thorax und Abdomen. In: *Fortschr Röntgenstr* 176 (2004), Nr. 7, S. 1021–30

[18] Hundt, W. ; Rust, F. ; Stabler, A. ; Wolff, H. ; Suess, C. ; Reiser, M. : Dose reduction in multislice computed tomography. In: *J Comput Assist Tomogr* 29 (2005), Nr. 1, S. 140–7

[19] Iannaccone, R. ; Laghi, A. ; Catalano, C. ; Mangiapane, F. ; Piacentini, F. ; Passariello, R. : Feasibility of ultra-low-dose multislice CT colonography for the detection of colorectal lesions: preliminary experience. In: *Eur Radiol* 13 (2003), Nr. 6, S. 1297–302

[20] Imhof, H. ; Schibany, N. ; Ba-Ssalamah, A. ; Czerny, C. ; Hojreh, A. ; Kainberger, F. ; Krestan, C. ; Kudler, H. ; Nobauer, I. ; Nowotny, R. : Spiral CT and radiation dose. In: *Eur J Radiol* 47 (2003), Nr. 1, S. 29–37

[21] Kachelriess, M. ; Watzke, O. ; Kalender, W. A.: Generalized multi-dimensional adaptive filtering for conventional and spiral single-slice, multi-slice, and cone-beam CT. In: *Med Phys* 28 (2001), Nr. 4, S. 475–90

[22] Kalender, W. A.: Grundlagen und Technik der Spiral-CT. In: *Radiologe* 39 (1999), Nr. 9, S. 809–19; quiz 819

[23] Kalender, W. A. ; Wolf, H. ; Suess, C. ; Gies, M. ; Greess, H. ; Bautz, W. A.: Dose reduction in CT by on-line tube current control: principles and validation on phantoms and cadavers. In: *Eur Radiol* 9 (1999), Nr. 2, S. 323–28

[24] Kalra, M. K. ; Maher, M. M. ; Rizzo, S. ; Saini, S. : Radiation exposure and projected risks with multidetector-row computed tomography scanning: clinical strategies and technologic developments for dose reduction. In: *J Comput Assist Tomogr* 28 Suppl 1 (2004), S. S46–9

[25] Kalra, M. K. ; Maher, M. M. ; Saini, S. : Multislice CT: update on radiation and screening. In: *Eur Radiol* 13 Suppl 5 (2003), S. M129–33

[26] Kalra, M. K. ; Prasad, S. ; Saini, S. ; Blake, M. A. ; Varghese, J. ; Halpern, E. F. ; Thrall, J. H. ; Rhea, J. T.: Clinical comparison of standard-dose and 50% reduced-dose abdominal CT: effect on image quality. In: *AJR Am J Roentgenol* 179 (2002), Nr. 5, S. 1101–6

[27] Kalra, M. K. ; Wittram, C. ; Maher, M. M. ; Sharma, A. ; Avinash, G. B. ; Karau, K. ; Toth, T. L. ; Halpern, E. ; Saini, S. ; Shepard, J. A.: Can noise reduction filters improve low-radiation-dose chest CT images? Pilot study. In: *Radiology* 228 (2003), Nr. 1, S. 257–64

[28] Keller, P. J. ; Drayer, B. P. ; Fram, E. K. ; Williams, K. D. ; Dumoulin, C. L. ; Souza, S. P.: MR angiography with two-dimensional acquisition and three-dimensional display. Work in progress. In: *Radiology* 173 (1989), Nr. 2, S. 527–32

[29] Keselbrener, L. ; Shimoni, Y. ; Akselrod, S. : Nonlinear filters applied on computerized axial tomography: theory and phantom images. In: *Med Phys* 19 (1992), Nr. 4, S. 1057–64

[30] Klüner, C. ; Hein, P. A. ; Gralla, O. ; Hein, E. ; Hamm, B. ; Romano, V. ; Rogalla, P. : Does ultra-low-dose CT with a radiation dose equivalent to that of KUB suffice to detect renal and ureteral calculi? In: *J Comput Assist Tomogr* (In Press)

[31] Kojima, A. ; Takeshita, K. ; Fukuchi, Y. ; Dambara, T. : Usefulness of low-dose single-slice CT (SSCT) for mass screening to detect various thoracic diseases including lung cancer. In: *Nihon Kokyuki Gakkai Zasshi* 42 (2004), Nr. 10, S. 887–92

[32] Krummenauer, F. : IV: Signifikanztests - wann welchen? In: *Klin Monatsbl Augenheilkd* 219 (2002), Nr. 11, S. 817–20

[33] Laub, G. A. ; Kaiser, W. A.: MR angiography with gradient motion refocusing. In: *J Comput Assist Tomogr* 12 (1988), Nr. 3, S. 377–82

[34] Marten, K. ; Funke, M. ; Obenauer, S. ; Baum, F. ; Grabbe, E. : Stellenwert unterschiedlicher Nachverarbeitungsverfahren in der Mehrschicht-Spiral-CT der akuten Lungenembolie. In: *Fortschr Röntgenstr* 175 (2003), Nr. 5, S. 635–9

[35] Mullins, M. E. ; Lev, M. H. ; Bove, P. ; O'Reilly, C. E. ; Saini, S. ; Rhea, J. T. ; Thrall, J. H. ; Hunter, G. J. ; Hamberg, L. M. ; Gonzalez, R. G.: Comparison of image quality between conventional and low-dose nonenhanced head CT. In: *AJNR Am J Neuroradiol* 25 (2004), Nr. 4, S. 533–8

[36] Nagel, H. D. ; Blobel, J. ; Brix, G. ; Ewen, K. ; Galanski, M. ; Hofs, P. ; Loose, R. ; Prokop, M. ; Schneider, K. ; Stamm, G. ; Stender, H. S. ; Suss, C. ; Turkay, S. ; Vogel, H. ; Wucherer, M. : 5 Jahre Konzentrierte Aktion Dosisreduktion CT - was wurde erreicht, was ist noch zu tun? In: *Fortschr Röntgenstr* 176 (2004), Nr. 11, S. 1683–94

[37] Napel, S. ; Marks, M. P. ; Rubin, G. D. ; Dake, M. D. ; McDonnell, C. H. ; Song, S. M. ; Enzmann, D. R. ; Jeffrey, J. : CT angiography with spiral CT and maximum intensity projection. In: *Radiology* 185 (1992), Nr. 2, S. 607–10

[38] Napel, S. ; Rubin, G. D. ; Jeffrey, J. : STS-MIP: a new reconstruction technique for CT of the chest. In: *J Comput Assist Tomogr* 17 (1993), Nr. 5, S. 832–8

[39] Napel, S. ; Bergin, C. ; Paranjpe, D. ; Rubin, G. D.: Maximum and Minimum Intensity Projection of Spiral CT Data for Simultaneous 3D Imaging of the Pulmonary Vasculature and Airways (Abstract). In: *78th*

Scientific Sessions Radiological Society of North America Bd. 185(p). Chicago : Radiology, 1992, S. 126

[40] Nitta, N. ; Takahashi, M. ; Murata, K. ; Morita, R. : Ultra low-dose helical CT of the chest: evaluation in clinical cases. In: *Radiat Med* 17 (1999), Nr. 1, S. 1–7

[41] Ohnesorge, B. ; Flohr, T. ; Schaller, S. ; Klingenbeck-Regn, K. ; Becker, C. ; Schopf, U. J. ; Bruning, R. ; Reiser, M. F.: Technische Grundlagen und Anwendungen der Mehrschicht-CT. In: *Radiologe* 39 (1999), Nr. 11, S. 923–31

[42] Parmar, M. S.: Kidney stones. In: *Bmj* 328 (2004), Nr. 7453, S. 1420–4

[43] Prasad, S. R. ; Wittram, C. ; Shepard, J. A. ; McLoud, T. ; Rhea, J. : Standard-dose and 50%-reduced-dose chest CT: comparing the effect on image quality. In: *AJR Am J Roentgenol* 179 (2002), Nr. 2, S. 461–5

[44] Prokop, M. : Überblick über Strahlendosis und Bildqualität in der Computertomographie. In: *Fortschr Röntgenstr* 174 (2002), Nr. 5, S. 631–6

[45] Prokop, M. : General principles of MDCT. In: *Eur J Radiol* 45 Suppl 1 (2003), S. S4–10

[46] Prokop, M. ; Shin, H. O. ; Schanz, A. ; Schaefer-Prokop, C. M.: Use of maximum intensity projections in CT angiography: a basic review. In: *Radiographics* 17 (1997), Nr. 2, S. 433–51

[47] Ripolles, T. ; Agramunt, M. ; Errando, J. ; Martinez, M. J. ; Coronel, B. ; Morales, M. : Suspected ureteral colic: plain film and sonography vs unenhanced helical CT. A prospective study in 66 patients. In: *Eur Radiol* 14 (2004), Nr. 1, S. 129–36

[48] Rogalla, P. ; Klüner, C. ; Taupitz, M. : Ultra-Niedrigdosis-CT zur Steinsuche in Nieren und ableitenden Harnwegen. In: *Aktuelle Urol* 35 (2004), Nr. 4, S. 307–9

[49] Rogalla, P. ; Stöver, B. ; Scheer, I. ; Juran, R. ; Gaedicke, G. ; Hamm, B. : Low-dose spiral CT: applicability to paediatric chest imaging. In: *Pediatr Radiol* 29 (1999), Nr. 8, S. 565–9

[50] Romano, V. ; Zaspel, U. ; Hein, P. ; Elgeti, T. ; Hamm, B. ; Rogalla, P. : MSCT Performed at a Chest Radiograph Dose: Detection and Characterization of Pulmonary Nodules Compared with Standard-dose CT (Abstract). In: *RSNA*. Chicago : Radiology, 2004

[51] Rubin, G. D.: Data explosion: the challenge of multidetector-row CT. In: *Eur J Radiol* 36 (2000), Nr. 2, S. 74–80

[52] Sasaki, T. ; Hanari, T. ; Sasaki, M. ; Oikawa, H. ; Gakumazawa, H. ; Okumura, M. ; Ikeda, Y. ; Toyoshima, N. : Reduction of radiation exposure in CT perfusion study using a quantum de-noising filter (Abstract). In: *Nippon Hoshasen Gijutsu Gakkai Zasshi* 60 (2004), Nr. 12, S. 1688–93

[53] Stöver, B. ; Rogalla, P. : CT-Untersuchungen beim Kind. In: *Radiologe* 39 (1999), Nr. 6, S. 455–62

[54] Vogt, C. ; Cohnen, M. ; Beck, A. ; vom Dahl, S. ; Aurich, V. ; Modder, U. ; Haussinger, D. : Detection of colorectal polyps by multislice CT colonography with ultra-low-dose technique: comparison with high-resolution videocolonoscopy. In: *Gastrointest Endosc* 60 (2004), Nr. 2, S. 201–9

[55] Wedegartner, U. ; Lorenzen, M. ; Lorenzen, J. ; Nolte-Ernsting, C. ; Weber, C. ; Dieckmann, C. ; Cramer, M. ; Schoder, V. ; Adam, G. : Mehrzeilen-Spiral CT (MSCT) des Beckenskelettes: Dosisoptimierung unter Berücksichtigung der Bildqualität. In: *Fortschr Röntgenstr* 176 (2004), Nr. 1, S. 106–12

[56] Wedegartner, U. ; Lorenzen, M. ; Nagel, H. D. ; Koops, A. ; Weber, C. ; Nolte-Ernsting, C. ; Schoder, V. ; Adam, G. : Dünn- und

dickschichtige MSCT-Untersuchungen bei Niedrigkontrastobjekten (Leberläsionen): Vergleich der Bildqualität bei gleicher Dosis. In: *Fortschr Röntgenstr* 176 (2004), Nr. 11, S. 1676–82

[57] Zou, Y. ; Sidky, E. Y. ; Pan, X. : Partial volume and aliasing artefacts in helical cone-beam CT. In: *Phys Med Biol* 49 (2004), Nr. 11, S. 2365–75

Anhang A

DicomProjector

Hier ist nur der Quelltext der Hauptklasse DcmVolume abgedruckt.

A.1 C++ - Header

Zunächst der C++ - Header `dcmvolume.h`:

```
1  #ifndef DCMVOLUME_H
2  #define DCMVOLUME_H
3
4
5  #include <vector>
6
7  #include "dicomdecoder.h"
8  #include "labelimg.h"
9
10 #define RENDER_XY 2
11 #define RENDER_XZ 1
12 #define RENDER_YZ 0
13 #define DIMENSION_X RENDER_YZ
14 #define DIMENSION_Y RENDER_XZ
15 #define DIMENSION_Z RENDER_XY
16
17 #define FILTER_PREVIEW 0
18 #define FILTER_VIEW 1
19
20
21 class DcmSlice {
22 protected:
23   std::vector< std::vector<short int> > pixels;
24 public:
25   const std::vector< short int >& operator[] (int i) const { return pixels[i]; }
26   std::vector< std::vector< short int > >& getPixels(void) { return pixels; }
27   const std::vector< std::vector< short int > >& getPixelsConst(void)
28           const { return pixels; }
29   int size(void) { return pixels.size(); }
30   friend ostream& operator <<(ostream &o, const DcmSlice &s);
31   friend istream& operator >>(istream &i, DcmSlice &s);
32   bool operator==(const DcmSlice& x) {
33       return this->zpos==x.zpos;
34   }
```

```
35    bool operator<(const DcmSlice& x) {
36        return this->zpos > x.zpos;
37    }
38    double xpos;
39    double ypos;
40    double zpos;
41    double xspacing;
42    double yspacing;
43    double RescaleIntercept;
44    double RescaleSlope;
45    double WindowCenter;
46    double WindowWidth;
47    int rows;
48    int cols;
49  };
50
51
52  class DcmVolume {
53  public:
54    DcmVolume(const SeriesInf& ser, int loadall=1);
55
56    void branding(LabelImg &label);
57    int blend(const DcmVolume& v2, double blendval=0.5);
58    void projectVAR(DcmVolume& dest,double tn, double ol, double mint, double maxt,
59                    const std::vector< double > weights,
60                    int renderPlane=RENDER_XY,int adjustHU=1) const;
61    void projectMIP2AVG(DcmVolume& dest, double slstart, double slend, int numslices,
62                    int renderPlane=RENDER_XY,
63                    LabelImg &label = LabelImg(NULL,0,0)) const;
64    void projectMIP2AVGmix(DcmVolume& dest, double slstart, double slend, int numslices,
65                    int renderPlane=RENDER_XY,
66                    LabelImg &label = LabelImg(NULL,0,0)) const;
67
68
69    DcmVolume(const DcmVolume& source, int xsize, int ysize, int zsize);
70    void load(string &fname);
71    DcmVolume() {}
72    const DcmSlice& operator[] (int i) { return volume[i]; }
73    int getNextSlice(void);
74    void save(const string &fname) const;
75    void render(int xsz, int ysz, double xoffset, double yoffset, double zoom,
76                    int rendermode, int filter, std::vector<unsigned char> &data) const;
77    short int getVoxel(double x, double y, double z, int filter) const;
78    short int getVoxel(double x, double y, int z, int rendermode, int filter) const;
79    void getVoxelSlice(vector <short int>& v, double x, double y, double z,
80                    double gagestep, int steps, int rendermode, int filter) const;
81    int setWindow(int ws, int wc);
82    inline int setPos(int i, double p) {if (pos[i]==p) return 0; pos[i]=p;return 1;}
83    inline int setGage(double g) {if (gage==g) return 0; gage=g;return 1;}
84    inline short int getIVoxel(int x, int y, int z) const
85          {if ((x>=0)&&(x<width[0])&&(y>=0)&&(y<width[1])&&(z>=0)&&(z<width[2]))
86                  return volume[z][y][x]; else return outside_value;}
87    inline void getWindowingParams(int &wf, int &ws, int &wu, int &wl,
88                    int pwsize=0xFFFFFFFF, int pwcenter=0xFFFFFFFF) const {
89      if (pwsize == 0xFFFFFFFF && pwcenter == 0xFFFFFFFF) {
90        pwsize = wsize; pwcenter = wcenter;
91      }
92        wf = int(65536.0*rescaleSlope/pwsize);
93        ws = int((rescaleIntercept+(pwsize>>1)-pwcenter)*65536/pwsize);
94        wu = (255*256 - ws)/wf;
95        wl = -(ws-256)/wf;
96    }
97    stringMap& getDicomData(void) {return dicomData;}
98    const stringMap& getDicomDataConst(void) const {return dicomData;}
99
100
101
```

```
102     inline short int getIVoxel(double px, double py, int pz, int rendermode) const {
103       int x; int y; int z;
104       if (rendermode == RENDER_XY) {
105               x=int(px/spacing[0]);y=int(py/spacing[1]);z=pz;}
106       else { if (rendermode == RENDER_XZ) {
107               x=int(px/spacing[0]);y=pz;z=int(py/spacing[2]);}
108         else { if (rendermode == RENDER_YZ) {
109                 x=pz;y=int(px/spacing[1]);z=int(py/spacing[2]);}
110           else return outside_value;
111         }
112       }
113       if ((x>=0)&&(x<width[0])&&(y>=0)&&(y<width[1])&&(z>=0)&&(z<width[2]))
114         return volume[z][y][x];
115       else return outside_value;
116     }
117     inline short int getIVoxel(int px, int py, int pz, int rendermode) const {
118       int x; int y; int z;
119       if (rendermode == RENDER_XY) {x=px;y=py;z=pz;}
120       else { if (rendermode == RENDER_XZ) {x=px;y=pz;z=py;}
121         else { if (rendermode == RENDER_YZ) {x=pz;y=px;z=py;}
122           else return outside_value;
123         }
124       }
125       if ((x>=0)&&(x<width[0])&&(y>=0)&&(y<width[1])&&(z>=0)&&(z<width[2]))
126               return volume[z][y][x]; else return outside_value;
127     }
128
129     inline double getSpacing(int i) const {return spacing[i];}
130     inline double getRealWidth(int i) const {return width[i]*spacing[i];}
131     inline double getPos(int i) const {return pos[i];}
132     inline double getGage(void) const {return gage;}
133     inline void getWindow(int &ws, int &wc) const {ws=wsize;wc=wcenter;}
134     inline int getWindowSize(void) const {return wsize;}
135     inline int getWindowCenter(void) const {return wcenter;}
136     inline int getRows(void) const {return width[0];}
137     inline int getCols(void) const {return width[1];}
138     inline int getNumSlices(void) const {return width[2];}
139     inline int getWidth(int i) const {return width[i];}
140     void analyseHisto(int &offset, int &swidth,double hwidth,
141                       int rendermode=RENDER_XY) const;
142     void getHisto(std::vector<unsigned int> &histo, int rendermode=RENDER_XY) const;
143     void adjustHisto(int soffset, int sswidth, int doffset, int dswidth);
144
145   protected:
146     std::vector< DcmSlice > volume;
147     std::vector<string>::const_iterator iflistpos;
148     std::vector<string>::const_iterator iflistend;
149     int width[3];
150     double spacing[3];
151     double pos[3];
152     double rescaleIntercept;
153     double rescaleSlope;
154     double gage;
155     int wsize;
156     int wcenter;
157     short int outside_value;
158     stringMap dicomData;
159
160     void DcmVolume::project(double zstart,double zend,double mint,
161                       double maxt,std::vector< std::vector< short int > >& data,
162                       int renderX=DIMENSION_X, int renderY=DIMENSION_Y,
163                       int renderPlane = RENDER_XY) const;
164     void DcmVolume::project(double zstart,double zend,const std::vector< double > weights,
165                       std::vector< std::vector< short int > >& data,
166                       int renderX, int renderY, int renderPlane) const;
167     void DcmVolume::projectMIP(double zstart,double zend,
168                       std::vector< std::vector< short int > >& data,
```

```
169                     int renderX=DIMENSION_X, int renderY=DIMENSION_Y,
170                     int renderPlane = RENDER_XY) const;
171     void DcmVolume::projectAVG(double zstart,double zend,
172                     std::vector< std::vector< short int > >& data,
173                     int renderX=DIMENSION_X, int renderY=DIMENSION_Y,
174                     int renderPlane = RENDER_XY) const;
175     void DcmVolume::getValues(DcmSlice& sl, const stringMap& m);
176     void dofinalstuff(void);
177   };
178
179
180
181
182   #endif
```

A.2 C++ - Body

Hier folgt der Quelltext der Programmdatei `dcmvolume.cxx`:

```
1   #include "dcmvolume.h"
2   #include <stdexcept>
3   #include <iostream>
4   #include <strstream>
5   #include <algorithm>
6   #include "tostring.h"
7
8   #define histo_maxsize 65536
9   #define histo_shift 32768
10
11
12  DcmVolume::DcmVolume(const SeriesInf& ser, int loadall):wsize(350),wcenter(50),gage(0.0) {
13    width[0]=0;width[1]=0;width[2]=0;
14    volume.resize(ser.getfilelist().size());
15    iflistpos = ser.getfilelist().begin();
16    iflistend = ser.getfilelist().end();
17    if (loadall)
18      while(getNextSlice()) {};
19  }
20
21  void DcmVolume::projectMIP2AVG(DcmVolume& dest, double slstart, double slend,
22                  int numslices, int renderPlane, LabelImg &label) const {
23    dest.dicomData = dicomData;
24
25    dest.rescaleIntercept =rescaleIntercept;
26    dest.rescaleSlope =rescaleSlope;
27    dest.gage =gage;
28    dest.wsize =wsize;
29    dest.wcenter =wcenter;
30    dest.outside_value =outside_value;
31
32
33    int renderX;
34    int renderY;
35    if (renderPlane == RENDER_XY) {
36      renderX = DIMENSION_X; renderY = DIMENSION_Y;
37    } else if (renderPlane == RENDER_XZ) {
38      renderX = DIMENSION_X; renderY = DIMENSION_Z;
39    } else if (renderPlane == RENDER_YZ) {
40      renderX = DIMENSION_Y; renderY = DIMENSION_Z;
41    }
42    dest.width[0]=width[renderX]; dest.width[1]=width[renderY]; dest.width[2]=numslices;
43    dest.spacing[0]=spacing[renderX]; dest.spacing[1]=spacing[renderY];
44    dest.spacing[2]=slend - slstart;
```

```
45     dest.pos[0]=pos[renderX]; dest.pos[1]=pos[renderY]; dest.pos[2]=pos[renderPlane];
46
47
48
49     dest.volume.resize(numslices);
50   cerr << "rendering " << numslices << " slices ";
51     double mint = 0.00000001;
52     double maxt = 0.99999999;
53     double adder = (maxt - mint) / (numslices-1);
54     for(int slice=0;slice<numslices;++slice) {
55       dest.volume[slice].xpos = 0;
56       dest.volume[slice].ypos = 0;
57       dest.volume[slice].zpos = dest.pos[2] + dest.spacing[2]*slice;
58       dest.volume[slice].xspacing = dest.spacing[0];
59       dest.volume[slice].yspacing = dest.spacing[1];
60       dest.volume[slice].RescaleIntercept = dest.rescaleIntercept;
61       dest.volume[slice].RescaleSlope = dest.rescaleSlope;
62       dest.volume[slice].WindowCenter = dest.wcenter;
63       dest.volume[slice].WindowWidth = dest.wsize;
64       dest.volume[slice].cols = dest.width[0];
65       dest.volume[slice].rows = dest.width[1];
66       project(slstart,slend,mint,maxt,dest.volume[slice].getPixels(),renderX,renderY,renderPlane);
67       mint += adder;
68   cerr << ".";
69     }
70   cerr << "finished" << endl;
71   }
72
73   void DcmVolume::projectMIP2AVGmix(DcmVolume& dest, double slstart, double slend,
74                   int numslices, int renderPlane, LabelImg &label) const {
75     dest.dicomData = dicomData;
76
77     dest.rescaleIntercept =rescaleIntercept;
78     dest.rescaleSlope =rescaleSlope;
79     dest.gage =gage;
80     dest.wsize =wsize;
81     dest.wcenter =wcenter;
82     dest.outside_value =outside_value;
83
84
85     int renderX;
86     int renderY;
87     if (renderPlane == RENDER_XY) {
88       renderX = DIMENSION_X; renderY = DIMENSION_Y;
89     } else if (renderPlane == RENDER_XZ) {
90       renderX = DIMENSION_X; renderY = DIMENSION_Z;
91     } else if (renderPlane == RENDER_YZ) {
92       renderX = DIMENSION_Y; renderY = DIMENSION_Z;
93     }
94     dest.width[0]=width[renderX]; dest.width[1]=width[renderY]; dest.width[2]=numslices;
95     dest.spacing[0]=spacing[renderX]; dest.spacing[1]=spacing[renderY];
96     dest.spacing[2]=slend - slstart;
97     dest.pos[0]=pos[renderX]; dest.pos[1]=pos[renderY]; dest.pos[2]=pos[renderPlane];
98
99
100
101    dest.volume.resize(numslices);
102  cerr << "rendering " << numslices << " slices ";
103    double mint = 0.00000001;
104    double maxt = 0.99999999;
105    double adder = (maxt - mint) / (numslices-1);
106
107    std::vector< std::vector<short int> > mip;
108    std::vector< std::vector<short int> > avg;
109    projectMIP(slstart,slend,mip,renderX,renderY,renderPlane);
110    projectAVG(slstart,slend,avg,renderX,renderY,renderPlane);
111
```

```
112     for(int slice=0;slice<numslices;++slice) {
113       dest.volume[slice].xpos = 0;
114       dest.volume[slice].ypos = 0;
115       dest.volume[slice].zpos = dest.pos[2] + dest.spacing[2]*slice;
116       dest.volume[slice].xspacing = dest.spacing[0];
117       dest.volume[slice].yspacing = dest.spacing[1];
118       dest.volume[slice].RescaleIntercept = dest.rescaleIntercept;
119       dest.volume[slice].RescaleSlope = dest.rescaleSlope;
120       dest.volume[slice].WindowCenter = dest.wcenter;
121       dest.volume[slice].WindowWidth = dest.wsize;
122       dest.volume[slice].cols = dest.width[0];
123       dest.volume[slice].rows = dest.width[1];
124
125       dest.volume[slice].getPixels().resize(dest.width[1]);
126       for(int y=0;y<dest.width[1];y++) {
127         dest.volume[slice].getPixels()[y].resize(dest.width[0]);
128         for(int x=0;x<dest.width[0];x++)
129             dest.volume[slice].getPixels()[y][x] =
130                     short int(mint * mip[y][x] + (1.0-mint) * avg[y][x]);
131       }
132       mint += adder;
133   cerr << ".";
134     }
135   cerr << "finished" << endl;
136   }
137
138   void DcmVolume::analyseHisto(int &offset, int &swidth,double hwidth,
139                   int rendermode) const {
140     vector<unsigned int> histo;
141     getHisto(histo,rendermode);
142     int sum=0;
143     for(unsigned int i=0;i<histo.size();i++) {
144       sum+=histo[i];
145     }
146     unsigned int tsum=0;
147     unsigned int start= int( sum * (1.0-hwidth)/2.0);
148     unsigned int end=sum-start;
149     int vstart,vend;
150     for(unsigned int i=0;i<histo.size();i++) {
151       if ((tsum < start) && ((tsum+histo[i]) >= start)) vstart=i;
152       if ((tsum < end) && ((tsum+histo[i]) >= end)) vend=i;
153       tsum+=histo[i];
154     }
155     offset=vstart - histo_shift;
156     swidth=vend-vstart;
157   }
158
159   void DcmVolume::branding(LabelImg &label)  {
160           int xd = width[0];
161           int yd = width[0];
162           if (label.getWidth() < xd) xd = label.getWidth();
163           if (label.getHeight() < yd) yd = label.getHeight();
164
165                   for(int x=0;x<xd;x++)
166                   for(int y=0;y<yd;y++)
167                           for(int z=0;z<width[2];z++) {
168                           int v = volume[z].getPixels()[y][x];
169                           v += label.getpix(x,y)*16;
170                           if (v>32767) v = 32767;
171                           volume[z].getPixels()[y][x] = v;
172                           }
173   }
174
175
176   void DcmVolume::projectVAR(DcmVolume& dest,double tn, double ol, double mint,
177                   double maxt, const std::vector<double> weights, int renderPlane,
178                   int adjustHU) const {
```

```
179     dest.dicomData = dicomData;
180     dest.rescaleIntercept =rescaleIntercept;
181     dest.rescaleSlope =rescaleSlope;
182     dest.gage =gage;
183     dest.wsize =wsize;
184     dest.wcenter =wcenter;
185     dest.outside_value =outside_value;
186
187     dest.getDicomData().insert(map<string, string>::value_type("ProjectedSliceThickness",
188                                toString(tn)));
189     string proj = "unknown";
190           if ((mint == 1.0)&&(maxt == 1.0))
191                   proj = "MIP";
192           else if ((mint == 0.0)&&(maxt == 1.0))
193                   proj = "AVG";
194           else {
195                   char p[100];
196                   if (weights.size()>0) {
197                           proj = "SoftMip:[";
198                           for(unsigned int i=0;i<weights.size();i+=2) {
199                                   if (i) proj += ",";
200                                   sprintf(p,"(%0.2f,%0.2f)",weights[i],weights[i+1]);
201                                   proj += p;
202                           }
203                           proj += "]";
204                   } else {
205                           sprintf(p,"SoftMip:[(0.00,0.00),(%0.2f,0.00),"
206                                           "(%0.2f,1.00),(%0.2f,1.00),"
207                                           "(%0.2f,0.00),(1.00,0.00)",
208                                           mint,mint,maxt,maxt);
209                           proj = toString(p);
210                   }
211           }
212     dest.getDicomData().insert(map<string, string>::value_type("Projection",proj));
213
214
215     int numslices = int((getRealWidth(renderPlane)-ol)/(tn-ol));
216     double scrap = getRealWidth(renderPlane) - ((tn-ol)*numslices + ol);
217     double zpos = scrap/2.0;
218
219     int renderX;
220     int renderY;
221     if (renderPlane == RENDER_XY) {
222       renderX = DIMENSION_X; renderY = DIMENSION_Y;
223     } else if (renderPlane == RENDER_XZ) {
224       renderX = DIMENSION_X; renderY = DIMENSION_Z;
225     } else if (renderPlane == RENDER_YZ) {
226       renderX = DIMENSION_Y; renderY = DIMENSION_Z;
227     }
228     dest.width[0]=width[renderX]; dest.width[1]=width[renderY]; dest.width[2]=numslices;
229     dest.spacing[0]=spacing[renderX]; dest.spacing[1]=spacing[renderY]; dest.spacing[2]=tn - ol;
230     dest.pos[0]=pos[renderX]; dest.pos[1]=pos[renderY]; dest.pos[2]=pos[renderPlane];
231
232
233
234     dest.volume.resize(numslices);
235   cerr << "rendering " << numslices << " slices ";
236     for(int slice=0;slice<numslices;++slice) {
237       dest.volume[slice].xpos = 0;
238       dest.volume[slice].ypos = 0;
239       dest.volume[slice].zpos = zpos + tn/2.0;
240       dest.volume[slice].xspacing = dest.spacing[0];
241       dest.volume[slice].yspacing = dest.spacing[1];
242       dest.volume[slice].RescaleIntercept = dest.rescaleIntercept;
243       dest.volume[slice].RescaleSlope = dest.rescaleSlope;
244       dest.volume[slice].WindowCenter = dest.wcenter;
245       dest.volume[slice].WindowWidth = dest.wsize;
```

```
246       dest.volume[slice].cols = dest.width[0];
247       dest.volume[slice].rows = dest.width[1];
248       if ((mint == 1.0)&&(maxt == 1.0))
249         projectMIP(zpos,zpos+tn,dest.volume[slice].getPixels(),renderX,renderY,renderPlane);
250       else if ((mint == 0.0)&&(maxt == 1.0))
251         projectAVG(zpos,zpos+tn,dest.volume[slice].getPixels(),renderX,renderY,renderPlane);
252           else {
253                   if (weights.size()>0)
254                           project(zpos,zpos+tn,weights,dest.volume[slice].getPixels(),
255                                   renderX,renderY,renderPlane);
256                   else
257                           project(zpos,zpos+tn,mint,maxt,dest.volume[slice].getPixels(),
258                                   renderX,renderY,renderPlane);
259
260           }
261
262   cerr << ".";
263       zpos+=tn-ol;
264     }
265   cerr << "finished" << endl;
266     if (adjustHU) {
267       cerr << "adjusting Histogram" << endl;
268       int soffset,sswidth;
269       analyseHisto(soffset,sswidth,0.50,renderPlane);
270       cerr << "source: offset="<<soffset<<" swidth="<<sswidth<<endl;
271       int doffset,dswidth;
272       dest.analyseHisto(doffset,dswidth,0.50,renderPlane);
273       dest.adjustHisto(soffset,sswidth,doffset,dswidth);
274     }
275   }
276
277   void DcmVolume::adjustHisto(int soffset, int sswidth, int doffset, int dswidth) {
278       double multi = 1.0*sswidth/dswidth;
279       cerr << "adjusting Histogram" << endl;
280       cerr << "multi="<<multi<<endl;
281       for(int x=0;x<width[0];x++)
282         for(int y=0;y<width[1];y++)
283           for(int z=0;z<width[2];z++) {
284             int v = volume[z].getPixels()[y][x];
285             volume[z].getPixels()[y][x] = int((v-doffset)*multi+soffset);
286           }
287   }
288
289
290   #define lower_int(x) ((int(x)<x)?int(x):(int(x)-1))
291
292   double zPixelAVG(const vector<DcmSlice> &volume, int x, int y, double z1, double z2) {
293     double value = 0.0;
294     if (int(z1) == lower_int(z2)) {
295       value = volume[int(z1)][y][x] * (z2 - z1);
296     } else {
297       for(int z = int(z1)+1; z < lower_int(z2); z++) value += volume[z][y][x];
298       value += volume[int(z1)][y][x] * (int(z1)+1-z1);
299       value += volume[lower_int(z2)][y][x] * (z2 - lower_int(z2));
300     }
301     return(value);
302   }
303
304   double yzPixelAVG(const vector<DcmSlice> &volume, int x,
305                     double y1, double y2, double z1, double z2) {
306     double value = 0.0;
307     if (int(y1) == lower_int(y2)) {
308       value = zPixelAVG(volume,x,int(y1),z1,z2) * (y2 - y1);
309     } else {
310       for(int y = int(y1)+1; y < lower_int(y2); y++) value += zPixelAVG(volume,x,y,z1,z2);
311       value += zPixelAVG(volume,x,int(y1),z1,z2) * (int(y1)+1-y1);
312       value += zPixelAVG(volume,x,lower_int(y2),z1,z2) * (y2 - lower_int(y2));
```

```
313     }
314     return(value);
315   }
316
317   double xyzPixelAVG(const vector<DcmSlice> &volume, double x1, double x2,
318                      double y1, double y2, double z1, double z2) {
319     double value = 0.0;
320     if (int(x1) == lower_int(x2)) {
321       value = yzPixelAVG(volume,int(x1),y1,y2,z1,z2) * (x2 - x1);
322     } else {
323       for(int x = int(x1)+1; x < lower_int(x2); x++)
324               value += yzPixelAVG(volume,x,y1,y2,z1,z2);
325       value += yzPixelAVG(volume,int(x1),y1,y2,z1,z2) * (int(x1)+1-x1);
326       value += yzPixelAVG(volume,lower_int(x2),y1,y2,z1,z2) * (x2 - lower_int(x2));
327     }
328     return(value);
329   }
330
331
332
333   DcmVolume::DcmVolume(const DcmVolume& source, int xsize, int ysize, int zsize)
334     :wsize(source.wsize),wcenter(source.wcenter),gage(source.gage)
335     ,rescaleIntercept(source.rescaleIntercept),rescaleSlope(source.rescaleSlope)
336     ,outside_value(source.outside_value),dicomData(source.dicomData) {
337
338     width[0]=xsize;
339     width[1]=ysize;
340     width[2]=zsize;
341
342     double pstep[3];
343     double ppos[3];
344     for(int i=0;i<3;i++) {
345       spacing[i] = source.spacing[i] * source.width[i] / width[i];
346       pos[i] = source.pos[i];
347       ppos[i] = 0.0;
348       pstep[i] = source.width[i] * 1.0 / width[i];
349     }
350
351     double normalizer = 1.0 / (pstep[0]*pstep[1]*pstep[2]);
352
353     volume.resize(width[2]);
354     for(int z=0;z<width[2];++z) {
355       volume[z].xpos = 0;
356       volume[z].ypos = 0;
357       volume[z].zpos = spacing[2]*z;
358       volume[z].xspacing = spacing[0];
359       volume[z].yspacing = spacing[1];
360       volume[z].RescaleIntercept = rescaleIntercept;
361       volume[z].RescaleSlope = rescaleSlope;
362       volume[z].WindowCenter = wcenter;
363       volume[z].WindowWidth = wsize;
364       volume[z].cols = width[0];
365       volume[z].rows = width[1];
366
367       ppos[1] = 0.0;
368       volume[z].getPixels().resize(width[1]);
369       for(int y=0;y<width[1];++y) {
370         ppos[0] = 0.0;
371         volume[z].getPixels()[y].resize(width[0]);
372         for(int x=0;x<width[0];++x) {
373           volume[z].getPixels()[y][x] = short int(xyzPixelAVG(source.volume,
374             ppos[0],ppos[0]+pstep[0],
375             ppos[1],ppos[1]+pstep[1],
376             ppos[2],ppos[2]+pstep[2]) * normalizer);
377           ppos[0] += pstep[0];
378         }
379         ppos[1] += pstep[1];
```

```
380       }
381       ppos[2] += pstep[2];
382     }
383   }
384
385   int DcmVolume::blend(const DcmVolume& v2, double blendval) {
386           if (width[0]!=v2.width[0]
387                   ||width[1]!=v2.width[1]
388                   ||width[2]!=v2.width[2]) return -1;
389           string p1 = getDicomData().find("Projection")->first;
390           string p2 = v2.getDicomDataConst().find("Projection")->first;
391           char t[100];
392           sprintf(t,",%0.2f]",blendval);
393           getDicomData().find("Projection")->second =
394                   string("Blend[")+p1+string(",")+p2+string(t);
395
396           double b2 = 1.0-blendval;
397           for(int z=0;z<width[2];++z) {
398                   for(int y=0;y<width[1];++y) {
399                           for(int x=0;x<width[0];++x) {
400                                   volume[z].getPixels()[y][x] =
401                                           short int(volume[z].getPixels()[y][x]* b2
402                                                   + v2.volume[z].getPixelsConst()[y][x]
403                                                   * blendval);
404                           }
405                   }
406           }
407
408
409   }
410
411
412   class c_pixelstack {
413   public:
414     c_pixelstack(int v, double l):val(v),length(l) {}
415     int val;
416     double length;
417     bool operator==(const c_pixelstack& x) {
418         return this->val==x.val;
419     }
420     bool operator<(const c_pixelstack& x) {
421         return this->val < x.val;
422     }
423   };
424
425   void DcmVolume::project(double zstart,double zend,double mint,double maxt,
426                   std::vector< std::vector< short int > >& data,
427                   int renderX, int renderY, int renderPlane) const {
428     zstart /= spacing[renderPlane]; // umrechnen in real-Pixelkoordinaten
429     zend /= spacing[renderPlane];
430     mint*=(zend-zstart); // adaptiere Thresholds an die Laenge des Pixelstacks
431     maxt*=(zend-zstart);
432     vector< c_pixelstack > pixelorder; // haelt die Indizes der Slices (mit Dicke)
433     // Anfang ist vielleicht ein Teilslice
434     pixelorder.push_back( c_pixelstack(int(zstart),int(zstart+1)-zstart));
435     // die ganzen Slices in der Mitte
436     for(int i=int(zstart+1);i<int(zend);++i) {
437       pixelorder.push_back( c_pixelstack(i,1.0) );
438     }
439     // Ende ist vielleicht ein Teilslice
440     pixelorder.push_back( c_pixelstack(int(zend),zend-int(zend)));
441
442     // der wirkliche Pixelstack
443     vector< c_pixelstack > pstack;
444     pstack.reserve(pixelorder.size());
445     // fuer jeden Bildpunkt des Slices
446     data.resize(width[renderY]);
```

```
447     for(int y=0;y<width[renderY];y++) {
448       data[y].resize(width[renderX]);
449       for(int x=0;x<width[renderX];x++) {
450         pstack.clear();
451         // Pixelstack fuellen
452         if (renderPlane == RENDER_XY)
453           for(vector< c_pixelstack >::iterator i=pixelorder.begin();i!=pixelorder.end();++i)
454             pstack.push_back( c_pixelstack(volume[i->val][y][x],i->length) );
455         else if (renderPlane == RENDER_XZ)
456           for(vector< c_pixelstack >::iterator i=pixelorder.begin();i!=pixelorder.end();++i)
457             pstack.push_back( c_pixelstack(volume[y][i->val][x],i->length) );
458         else if (renderPlane == RENDER_YZ)
459           for(vector< c_pixelstack >::iterator i=pixelorder.begin();i!=pixelorder.end();++i)
460             pstack.push_back( c_pixelstack(volume[y][x][i->val],i->length) );
461 
462         // Pixelstack sortieren
463         std::sort(pstack.begin(), pstack.end());
464 
465             vector< c_pixelstack >::iterator pstack_it = pstack.begin();
466         double stackpos = 0.0; // aktuelle Position im Stack
467         double value = 0.0; // Summierte Farbwerte
468         // Anfang suchen
469         while( (stackpos + pstack_it->length) <= mint) {
470           stackpos += pstack_it->length;
471           pstack_it++;
472         }
473         // ersten relevanten Pixel im Stack entsprechend zurechtschneiden
474         pstack_it->length -= mint - stackpos;
475         stackpos = mint;
476         // durch den Stack laufen und aufsummieren
477         while( (stackpos + pstack_it->length) < maxt) {
478           stackpos += pstack_it->length;
479           value += pstack_it->length * pstack_it->val;
480           pstack_it++;
481         }
482         // letzten, zurechtgeschnittenen Pixel dazu
483         value += (maxt - stackpos) * pstack_it->val;
484         // normalisieren und abspeichern
485         data[y][x] = short int (value / (maxt - mint));
486       }
487     }
488   }
489 
490 
491 
492 
493 
494   double getSegment(const std::vector< double > weights, double start,
495                     double maxend, double &value) {
496           double startx;
497           double starty;
498           double endx;
499           double endy;
500 
501       int numpoints = weights.size()>>1;
502           int i = 0;
503           while((weights[i*2]<=start) && (i<(numpoints-1))) {
504                   startx = weights[i*2];
505                   starty = weights[i*2+1];
506                   i++;
507           }
508           endx = weights[i*2];
509           endy = weights[i*2+1];
510 
511           if (endx>maxend) {
512                   endy = starty + (endy-starty) * (maxend-startx)/(endx-startx);
513                   endx = maxend;
```

```
514         }
515         if (startx<start) {
516                 starty = starty + (endy-starty) * (start-startx)/(endx-startx);
517         }
518
519         value = (double)((starty + endy)/2.0);
520         return(endx);
521 }
522
523 void generateOpacFunc(const std::vector< double > weights, int numPlanes,
524                 std::vector< int > &opacFunc) {
525         double plstart;
526         double newstart;
527         double plend;
528
529         double tval;
530         double val;
531         double sum;
532
533         sum=0;
534         vector<double> fOpacFunc;
535         for(int i=0;i<numPlanes;i++) {
536                 plstart = i*1.0/numPlanes;
537                 plend = (i+1.0)/numPlanes;
538                 val = 0.0;
539                 while(plstart<plend) {
540                         newstart = getSegment(weights,plstart,plend,tval);
541                         val += tval * (newstart-plstart);
542                         plstart = newstart;
543                 }
544                 fOpacFunc.push_back( val );
545                 sum += fOpacFunc[i];
546         }
547         for(i=0;i<numPlanes;i++) {
548                 double v = 1024.0/sum * fOpacFunc[i];
549                 double iv = (int) v;
550                 opacFunc.push_back( ((iv-v)<0.5)?((int)iv):((int)(iv)+1) );
551         }
552 }
553
554 int compare_short( const void *arg1, const void *arg2 )
555 {
556    return (*((short int *)arg1) - *((short int*)arg2 ));
557 }
558
559
560 void DcmVolume::project(double zstart,double zend,const std::vector< double > weights,
561                 std::vector< std::vector< short int > >& data,
562                 int renderX, int renderY, int renderPlane) const {
563   zstart /= spacing[renderPlane]; // umrechnen in real-Pixelkoordinaten
564   zend /= spacing[renderPlane];
565
566   int izstart = (int)zstart;
567   int izend = (int)zend;
568   if (izend < zend) izend++;
569   int numPlanes = izend-izstart;
570   std::vector< int > iweights;
571   generateOpacFunc(weights, numPlanes, iweights);
572
573   std::vector<short int> sorted_data;
574   sorted_data.resize(width[renderX]*width[renderY]*numPlanes);
575   int index = 0;
576   for(int y=0;y<width[renderY];y++) {
577     for(int x=0;x<width[renderX];x++) {
578                 for(int i=izstart; i < izend; i++)
579                         sorted_data[index++] = getIVoxel(x,y,i, renderPlane);
580         }
```

```
581     }
582     std::vector<short int>::iterator si_start = sorted_data.begin();
583     std::vector<short int>::iterator si_end = sorted_data.begin();
584     si_end+=numPlanes;
585     for(int y=0;y<width[renderY];y++) {
586       for(int x=0;x<width[renderX];x++) {
587                   std::sort(si_start, si_end);
588                   si_start+=numPlanes;
589                   si_end+=numPlanes;
590           }
591     }
592
593     data.resize(width[renderY]);
594     index = 0;
595     for(int y=0;y<width[renderY];y++) {
596       data[y].resize(width[renderX]);
597           for(int x=0;x<width[renderX];x++) {
598           int sum = 0;
599                   for(int i=0;i<numPlanes;i++)
600                           sum += sorted_data[index++]*iweights[i];
601                   data[y][x] = short int (sum / 1024);
602           }
603     }
604   }
605
606
607
608
609   void DcmVolume::projectMIP(double zstart,double zend,
610                   std::vector< std::vector< short int > >& data,
611                   int renderX, int renderY, int renderPlane) const {
612     zstart /= spacing[renderPlane]; // umrechnen in real-Pixelkoordinaten
613     zend /= spacing[renderPlane];
614     int izstart = int(zstart);
615     int izend = lower_int(zend) + 1;
616
617     data.resize(width[renderY]);
618     for(int y=0;y<width[renderY];y++) {
619       data[y].resize(width[renderX]);
620       for(int x=0;x<width[renderX];x++) {
621         short int val = -32768;
622         for(int i=izstart;i < izend;++i) {
623           val = max(val, getIVoxel(x,y,i, renderPlane));
624         }
625         data[y][x] = val;
626       }
627     }
628   }
629
630   void DcmVolume::projectAVG(double zstart,double zend,
631                   std::vector< std::vector< short int > >& data,
632                   int renderX, int renderY, int renderPlane) const {
633     zstart /= spacing[renderPlane]; // umrechnen in real-Pixelkoordinaten
634     zend /= spacing[renderPlane];
635     int izstart = int(zstart)+1;
636     int izend = int(zend);
637
638     data.resize(width[renderY]);
639     for(int y=0;y<width[renderY];y++) {
640       data[y].resize(width[renderX]);
641       for(int x=0;x<width[renderX];x++) {
642         if (int(zstart) != int (zend)) {
643           double val = 0;
644           for(int i=izstart;i < izend;++i)
645             val += getIVoxel(x,y,i, renderPlane);
646
647           if (zstart < izstart) val += getIVoxel(x,y,izstart - 1, renderPlane)
```

```
648                 * (izstart - zstart);
649         if (zend > izend) val += getIVoxel(x,y,izend, renderPlane) * (zend - izend);
650         data[y][x] = short int(val / (zend - zstart));
651
652       } else data[y][x] = getIVoxel(x,y,int(zstart), renderPlane);
653     }
654   }
655 }
656
657
658 template <class T>
659 bool binary_write(std::streambuf* sb, T const* t, std::size_t n=1)
660 {
661     return sb->sputn(
662         reinterpret_cast<char const*>(t),
663         n * sizeof *t
664     ) == n * sizeof *t;
665 }
666
667 template <class T>
668 bool binary_read(std::streambuf* sb, T *t, std::size_t n=1)
669 {
670     return sb->sgetn(
671         reinterpret_cast<char *>(t),
672         n * sizeof *t
673     ) == n * sizeof *t;
674 }
675
676 ostream& operator <<(ostream &o, const DcmSlice &s) {
677     std::streambuf* sb = o.rdbuf();
678     assert(sb);
679     for(int y=0;y<s.rows;++y)
680         binary_write(sb, &s.pixels[y][0], s.cols);
681     return o;
682 }
683
684 istream& operator >>(istream &i, DcmSlice &s) {
685     std::streambuf* sb = i.rdbuf();
686     assert(sb);
687     s.pixels.clear();
688     s.pixels.resize(s.rows);
689     for(int y=0;y<s.rows;++y) {
690       s.pixels[y].clear();
691       s.pixels[y].resize(s.cols);
692       binary_read(sb, &s.pixels[y][0], s.cols);
693     }
694     return i;
695 }
696
697
698 ostream& operator <<(ostream &o, const stringMap &map) {
699   std::streambuf* sb = o.rdbuf();
700   int msize = map.size();
701   binary_write(sb, &msize);
702   for(stringMap::const_iterator i = map.begin(); i != map.end(); ++i) {
703     string s = i->first;
704     if (s.length() > 255) s.erase(256);
705     unsigned char len = s.length();
706     binary_write(sb, &len);
707     o << s;
708     s = i->second;
709     if (s.length() > 255) s.erase(256);
710     len = s.length();
711     binary_write(sb, &len);
712     o << s;
713   }
714   return o;
```

```
715  }
716
717  istream& operator >>(istream &i, stringMap &map) {
718    std::streambuf* sb = i.rdbuf();
719    int num;
720    binary_read(sb,&num);
721    char key[256];
722    char val[256];
723    unsigned char keyl;
724    unsigned char vall;
725    for(int c=0; c != num; ++c) {
726      binary_read(sb,&keyl);
727      binary_read(sb,key,keyl);
728      binary_read(sb,&vall);
729      binary_read(sb,val,vall);
730      key[keyl]=0;
731      val[vall]=0;
732      map.insert(stringMap::value_type(key,val));
733    }
734    return i;
735  }
736
737
738  void DcmVolume::load(string &fname) {
739    std::ifstream infile(fname.c_str(), ios::in|ios::binary);
740    if (! infile.is_open())
741      throw(runtime_error("Error reading " + fname));
742    std::streambuf* sb = infile.rdbuf();
743      binary_read(sb, &width[0],3);
744      binary_read(sb, &spacing[0],3);
745      binary_read(sb, &pos[0],3);
746      binary_read(sb, &rescaleIntercept);
747      binary_read(sb, &rescaleSlope);
748      binary_read(sb, &wsize);
749      binary_read(sb, &wcenter);
750      binary_read(sb, &outside_value);
751      dicomData.clear();
752      infile >> dicomData;
753    cerr<<"reading "<<width[DIMENSION_Z]<<" frames ";
754    volume.clear();
755    volume.reserve(width[DIMENSION_Z]);
756    for(int i=0;i<width[DIMENSION_Z];++i) {
757        DcmSlice slice;
758        volume.push_back(slice);
759        volume[i].rows = width[1];
760        volume[i].cols = width[0];
761        infile >> volume[i];
762        cerr<<".";
763    }
764    cerr<<" finished!"<<endl;
765  }
766
767
768
769
770
771
772  void DcmVolume::save(const string &fname) const {
773    std::ofstream savefile(fname.c_str(), ios::out|ios::binary);
774
775    {
776      if (! savefile.is_open())
777        throw(runtime_error("Error creating " + fname));
778      std::streambuf* sb = savefile.rdbuf();
779      binary_write(sb, &width[0],3);
780      binary_write(sb, &spacing[0],3);
781      binary_write(sb, &pos[0],3);
```

```
782       binary_write(sb, &rescaleIntercept);
783       binary_write(sb, &rescaleSlope);
784       binary_write(sb, &wsize);
785       binary_write(sb, &wcenter);
786       binary_write(sb, &outside_value);
787       savefile << dicomData;
788     cerr<<"Saving "<<width[DIMENSION_Z]<<" frames ";
789       for(int i=0;i<width[DIMENSION_Z];++i) {
790         savefile << volume[i];
791     cerr<<".";
792       }
793     }
794     cerr<<" finished!"<<endl;
795   }
796
797   int DcmVolume::setWindow(int ws, int wc) {
798     if (ws<2) ws=2;
799     if (ws>4095) ws=4095;
800     if (wc<-2048) wc=-2048;
801     if (wc>2047) wc=2047;
802     if ((ws!=wsize)||(wc!=wcenter)) {
803       wsize=ws;
804       wcenter=wc;
805       return(1);
806     }
807     return(0);
808   }
809
810   void DcmVolume::render(int xsz, int ysz, double xoffset, double yoffset,
811                          double zoom, int rendermode, int filter,
812                          std::vector<unsigned char> &data) const {
813     if (!xsz || !ysz || !width[0] || !width[1] || !width[2]) return;
814
815     int xind = 0; int yind = 1; int slind = 2;
816     int slpos = int(pos[2]/spacing[2]);
817     if (rendermode == RENDER_XZ) {
818             yind = 2; slind = 1; slpos = int(pos[1]/spacing[1]);}
819     if (rendermode == RENDER_YZ) {
820             xind = 1; yind = 2; slind = 0; slpos = int(pos[0]/spacing[0]);}
821
822     double xwidth = width[xind] * spacing[xind] * zoom;
823     double xstart = (xwidth - xsz)/2.0;
824     double xend = xwidth - xstart;
825     xstart /= zoom;
826     xend /= zoom;
827     xstart += xoffset;
828     xend += xoffset;
829     double xadd = (xend - xstart)/xsz;
830
831     double ywidth = width[yind] * spacing[yind] * zoom;
832     double ystart = (ywidth - ysz)/2.0;
833     double yend = ywidth - ystart;
834     ystart /= zoom;
835     yend /= zoom;
836     ystart += yoffset;
837     yend += yoffset;
838     double yadd = (yend - ystart)/ysz;
839
840     double mx = width[xind]*spacing[xind];
841     double my = width[yind]*spacing[yind];
842
843     int off=0;
844     double yp = ystart;
845
846     int windowfactor;
847     int windowshifter;
848     int wupper;
```

```
849     int wlower;
850     getWindowingParams(windowfactor,windowshifter,wupper,wlower);
851
852     int ot = outside_value;
853     if (ot<wlower) ot=wlower; if (ot>wupper) ot=wupper;
854     ot = (ot*windowfactor + windowshifter)>>8;
855
856     for(int ay=0;ay<ysz;ay++) {
857       double xp = xstart;
858       if ((yp>=0.0)&&(yp<=my)) {
859         for(int ax=0;ax<xsz;ax++) {
860           if ((xp>=0.0)&&(xp<=mx)) {
861             int t=0;
862             t = getVoxel(xp, yp, slpos, rendermode, filter);
863             if (t<wlower) t=wlower; if (t>wupper) t=wupper;
864             t = (t*windowfactor + windowshifter)>>8;
865             data[off++] = t;
866             data[off++] = t;
867             data[off++] = t;
868           } else {
869             data[off++] = ot;
870             data[off++] = ot;
871             data[off++] = ot;
872           }
873           xp += xadd;
874         }
875       } else { memset(&data[off],ot,xsz*3); off+=xsz*3;}
876       yp += yadd;
877     }
878   }
879
880   inline int interpolate(int v0, int v1, int v2, int v3, double x) {
881     return int(v1*(1-x) + v2*x);
882   }
883
884   short int DcmVolume::getVoxel(double x, double y, int z, int rendermode,
885                   int filter) const {
886     if (x<0 || y<0 || z<0
887       ||(rendermode == RENDER_XY && (x>width[0]*spacing[0]
888                       || y>width[1]*spacing[1] || z>width[2]))
889       ||(rendermode == RENDER_XZ && (x>width[0]*spacing[0]
890                       || y>width[2]*spacing[2] || z>width[1]))
891       ||(rendermode == RENDER_YZ && (x>width[1]*spacing[1]
892                       || y>width[2]*spacing[2] || z>width[0])) ) {
893         return outside_value;
894       }
895   if (filter == FILTER_PREVIEW) return getIVoxel(x,y,z,rendermode);
896     else {
897       if (rendermode == RENDER_XY) {x=x/spacing[0];y=y/spacing[1];}
898       else { if (rendermode == RENDER_XZ) {x=x/spacing[0];y=y/spacing[2];}
899         else { x=x/spacing[1];y=y/spacing[2];}}
900
901       int ix = int(x);
902       int iy = int(y);
903
904       int x0 = interpolate(0, getIVoxel(ix,iy,z), getIVoxel(ix,iy+1,z), 0, y-iy);
905       int x1 = interpolate(0, getIVoxel(ix+1,iy,z), getIVoxel(ix+1,iy+1,z), 0, y-iy);
906       return interpolate(0, x0, x1, 0, x-ix);
907     }
908   }
909
910
911
912   short int DcmVolume::getVoxel(double x, double y, double z, int filter) const {
913     if (x < 0.0 || x > (width[0]*spacing[0])
914       || y < 0.0 || y > (width[1]*spacing[1])
915       || z < 0.0 || z > (width[2]*spacing[2])) {
```

```
916         return outside_value;
917       }
918     x /= spacing[0];
919     y /= spacing[1];
920     z /= spacing[2];
921     if (filter == FILTER_PREVIEW) return getIVoxel(int(x)  ,int(y)  ,int(z));
922
923     int ix = int(x);
924     int iy = int(y);
925     int iz = int(z);
926     int y0x0 = interpolate(0,
927                     getIVoxel(ix  ,iy  ,iz), getIVoxel(ix  ,iy  ,iz+1), 0, z-iz);
928     int y0x1 = interpolate(0,
929                     getIVoxel(ix+1,iy  ,iz), getIVoxel(ix+1,iy  ,iz+1), 0, z-iz);
930     int y1x0 = interpolate(0,
931                     getIVoxel(ix  ,iy+1,iz), getIVoxel(ix  ,iy+1,iz+1), 0, z-iz);
932     int y1x1 = interpolate(0,
933                     getIVoxel(ix+1,iy+1,iz), getIVoxel(ix+1,iy+1,iz+1), 0, z-iz);
934
935     int x0 = interpolate(0, y0x0, y1x0, 0, y-iy);
936     int x1 = interpolate(0, y0x1, y1x1, 0, y-iy);
937
938     return interpolate(0, x0, x1, 0, x-ix);
939   }
940
941
942   int DcmVolume::getNextSlice(void) {
943     if (iflistpos != iflistend) {
944       DicomDecoder dd(*iflistpos);
945       DcmSlice slice;
946       stringMap metadata;
947       int t1,t2,t3,t4;
948       dd.getMetaData(metadata);
949       dd.getSlice(slice.getPixels(),slice.rows,slice.cols,t1,t2,t3,t4);
950       getValues(slice,metadata);
951       volume[width[2]]=slice;
952       outside_value = slice[slice.cols-1][slice.rows-1];
953       if (width[2]) {
954         if ((volume[width[2]].cols!=volume[width[2]-1].cols)
955           ||(volume[width[2]].rows!=volume[width[2]-1].rows))
956           throw std::runtime_error("different format of slices in one series");
957       } else dicomData = metadata;
958       ++width[2];
959       ++iflistpos;
960       if (iflistpos == iflistend) {
961         dofinalstuff();
962       }
963       cerr<<".";
964       return 1;
965     }
966     else return 0;
967   }
968
969   void DcmVolume::dofinalstuff(void) {
970     std::sort(volume.begin(), volume.end());
971     width[0]=volume[0].rows;
972     width[1]=volume[0].cols;
973     spacing[0]=volume[0].xspacing;
974     spacing[1]=volume[0].yspacing;
975     spacing[2]=volume[1].zpos-volume[0].zpos;
976     if (spacing[2]<0) spacing[2]=-spacing[2];
977     pos[0] = width[0]*spacing[0]/2.0;
978     pos[1] = width[1]*spacing[1]/2.0;
979     pos[2] = width[2]*spacing[2]/2.0;
980     rescaleIntercept = volume[0].RescaleIntercept;
981     rescaleSlope = volume[0].RescaleSlope;
982     if (rescaleSlope == 0.0) rescaleSlope = 1.0;
```

```
983     wsize = int(volume[0].WindowWidth);
984     if (wsize < 1) wsize = 2;
985     wcenter = int(volume[0].WindowCenter);
986   }
987
988
989   void DcmVolume::getValues(DcmSlice& sl, const stringMap& m) {
990     string s    = m.find("ImagePosition")->second;
991     char *t;
992     sl.xpos = strtod(s.c_str(),&t);
993     sl.ypos = strtod(t+1,&t);
994     sl.zpos = strtod(t+1,&t);
995     s = m.find("PixelSpacing")->second;
996     sl.xspacing = strtod(s.c_str(),&t);
997     sl.yspacing = strtod(t+1,&t);
998
999     s = m.find("RescaleIntercept")->second;
1000    sl.RescaleIntercept = strtod(s.c_str(),&t);
1001    s = m.find("RescaleSlope")->second;
1002    sl.RescaleSlope = strtod(s.c_str(),&t);
1003
1004    s = m.find("WindowCenter")->second;
1005    sl.WindowCenter = strtod(s.c_str(),&t);
1006    s = m.find("WindowWidth")->second;
1007    sl.WindowWidth = strtod(s.c_str(),&t);
1008  }
1009
1010  void DcmVolume::getHisto(std::vector<unsigned int> &histo, int rendermode) const {
1011    histo.resize(histo_maxsize);
1012    for(int i=0;i<histo_maxsize;i++) histo[i]=0;
1013
1014    double fxlow  = 0.13;
1015    double fxhigh = 0.40;
1016    double fylow  = 0.20;
1017    double fyhigh = 0.80;
1018    double fzlow  = 0.20;
1019    double fzhigh = 0.80;
1020    if (rendermode = RENDER_XZ) {
1021      double t;
1022      t = fylow; fylow = fzlow; fzlow = t;
1023      t = fyhigh; fyhigh = fzhigh; fzhigh = t;
1024    } else if (rendermode =  RENDER_YZ) {
1025      double t;
1026      t = fxlow; fxlow = fylow; fylow = fzlow; fzlow = t;
1027      t = fxhigh; fxhigh = fyhigh; fyhigh = fzhigh; fzhigh = t;
1028    }
1029    int xlow = int( width[0] * 0.13 );
1030    int xhigh = int( width[0] * 0.40 );
1031    int ylow = int( width[1] * 0.20 );
1032    int yhigh = int( width[1] * 0.80 );
1033    int zlow = int (width[2] * 0.20);
1034    int zhigh = int(width[2] * 0.80);
1035    for(int x=xlow;x<xhigh;x++)
1036      for(int y=ylow;y<yhigh;y++)
1037        for(int z=zlow;z<zhigh;z++) {
1038          int val=volume[z][y][x] + histo_shift;
1039          if ((val < 0)||(val >= histo_maxsize))
1040                  std::cerr << "volume::getHist val=" << val <<endl;
1041          else histo[val]++;
1042        }
1043  }
```

Anhang B

GradientFinder

```
1  require 'gtk2'
2  require 'csv'
3  require 'math/statistics'
4
5
6
7  MIP = 'mip'
8  AVG = 'avg'
9
10
11 LINESENS=7
12 V=0
13 H=1
14
15 LOWCUTOFF = 0.2
16 HIGHCUTOFF = 0.8
17
18 $csvoutname = "result"
19
20
21 class Array
22   include Math::Statistics
23 end
24
25 class Object
26   def dclone
27     Marshal::load(Marshal.dump(self))
28   end
29 end
30
31 def main
32   Gtk.init
33   if (ARGV.size>0)
34         pw = PhantomWidget.new()
35         ARGV.each do |j|
36                 pw.loadJobs(j)
37                 pw.executeJobs
38         end
39   else
40           gfWindow = GradientFinder.new()
41           gfWindow.show_all
42           Gtk.main
43   end
44 end
45
```

```
46  class GradientFinder < Gtk::Window
47    @Version = 0.01
48    @statusbar = nil
49    @@IF = Gtk::ItemFactory
50    @phantomW = nil
51    def initialize()
52      super(Gtk::Window::TOPLEVEL);
53      self.set_title( "GradientFinder" + @Version.to_s )
54      self.set_size_request( 300, 200 )
55      self.signal_connect( "delete_event" ) { Gtk.main_quit }
56      mainVBox = Gtk::VBox.new( false, 0 )
57      self.add( mainVBox )
58      mainVBox.pack_start( getMenuBar, false, false, 0)
59
60      frame = Gtk::Frame.new
61      frame.set_shadow_type( Gtk::SHADOW_IN )
62      @statusbar = Gtk::Statusbar.new
63      @phantomW = PhantomWidget.new(@statusbar)
64      frame.add( @phantomW )
65      mainVBox.pack_start( frame, true, true, 0)
66      mainVBox.pack_start( @statusbar, false, false, 0);
67    end
68
69    private
70
71    def onFileOpen
72      dialog = Gtk::FileChooserDialog.new("Open File",
73                                          self,
74                                          Gtk::FileChooser::ACTION_OPEN,
75                                          nil,
76                                          [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL],
77                                          [Gtk::Stock::OPEN, Gtk::Dialog::RESPONSE_ACCEPT])
78
79      if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
80        @phantomW.readImages(dialog.filename)
81        self.set_title( "GradientFinder" + @Version.to_s + "      "
82                  + File.basename(dialog.filename))
83      end
84      dialog.destroy
85    end
86    def onLoadJobs
87      dialog = Gtk::FileChooserDialog.new("Open File",
88                                          self,
89                                          Gtk::FileChooser::ACTION_OPEN,
90                                          nil,
91                                          [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL],
92                                          [Gtk::Stock::OPEN, Gtk::Dialog::RESPONSE_ACCEPT])
93
94      if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
95        @phantomW.loadJobs(dialog.filename)
96      end
97      dialog.destroy
98    end
99    def onSaveJobs
100     dialog = Gtk::FileChooserDialog.new("Save File",
101                                         self,
102                                         Gtk::FileChooser::ACTION_SAVE,
103                                         nil,
104                                         [Gtk::Stock::CANCEL, Gtk::Dialog::RESPONSE_CANCEL],
105                                         [Gtk::Stock::OPEN, Gtk::Dialog::RESPONSE_ACCEPT])
106
107     if dialog.run == Gtk::Dialog::RESPONSE_ACCEPT
108       @phantomW.saveJobs(dialog.filename)
109     end
110     dialog.destroy
111   end
112
```

```
113     def onProjection (p)
114       @phantomW.showProjection(p)
115     end
116
117     def onStoreCSV
118       @phantomW.storeCSV
119     end
120     def onAddJob
121       @phantomW.addJob
122     end
123     def onExecuteJobs
124       @phantomW.executeJobs
125     end
126     def onRefresh
127       @phantomW.onRefresh
128     end
129
130     def getMenuBar
131       print_selected = Proc.new {|data, w| puts "Hello World! with #{w}, #{data}"}
132       iF = Gtk::ItemFactory
133       menu_items = [
134         ["/_File"],
135         ["/File/_Open",     @@IF::STOCK_ITEM, "<CTRL>O", Gtk::Stock::OPEN,
136                   Proc.new {onFileOpen}, "Open"],
137         ["/File/_Store CSV",     @@IF::STOCK_ITEM, "<CTRL>S",
138                   Gtk::Stock::SAVE, Proc.new {onStoreCSV()} ,"Store"],
139         ["/File/sep1",      @@IF::SEPARATOR],
140         ["/File/_Quit",     @@IF::STOCK_ITEM, "<CTRL>Q",
141                   Gtk::Stock::QUIT, Proc.new {Gtk.main_quit}],
142         ["/_Projection",       @@IF::BRANCH],
143         ["/Projection/_MIP"      ,  @@IF::RADIO_ITEM , "M", nil,
144                   Proc.new {onProjection(MIP)}, 1],
145         ["/Projection/_AVG"      ,  "/Projection/MIP", "A", nil,
146                   Proc.new {onProjection(AVG)}, 3],
147         ["/Projection/_Refresh Image",     @@IF::STOCK_ITEM, "R",
148                   Gtk::Stock::REFRESH, Proc.new {onRefresh()} ,4],
149         ["/_Jobs",       @@IF::BRANCH],
150         ["/Jobs/_Add Job"         ,  @@IF::STOCK_ITEM, "J",
151                   Gtk::Stock::ADD, Proc.new {onAddJob()}, 1],
152         ["/Jobs/_Execute Jobs"      ,  @@IF::STOCK_ITEM, "E",
153                   Gtk::Stock::EXECUTE, Proc.new {onExecuteJobs()}, 2],
154         ["/Jobs/sep1",      @@IF::SEPARATOR],
155         ["/Jobs/_Load",     @@IF::STOCK_ITEM, "",
156                   Gtk::Stock::OPEN, Proc.new {onLoadJobs()}, "Load"],
157         ["/Jobs/_Save",     @@IF::STOCK_ITEM, "",
158                   Gtk::Stock::SAVE, Proc.new {onSaveJobs()} ,"Save"],
159         ["/_Help",        @@IF::LAST_BRANCH],
160         ["/_Help/About", @@IF::ITEM],
161       ]
162       accel_group = Gtk::AccelGroup.new
163       item_factory = Gtk::ItemFactory.new(Gtk::ItemFactory::TYPE_MENU_BAR,
164                   "<main>", accel_group)
165
166       item_factory.create_items(menu_items)
167       self.add_accel_group(accel_group)
168       item_factory.get_widget("<main>")
169     end
170
171   end
172
173   class PhantomWidget < Gtk::EventBox
174     def initialize(sbar=nil)
175       super()
176       @statusbar = sbar
177       @realpixbuf= {}
178       @pixbuf= {}
179       @imgData = {}
```

```
180       @fname = {}
181       @view = MIP
182       @borders = []
183       @wborders = []
184       @xres = nil
185       @yres = nil
186       @fact = []
187       @lineDrag = nil
188       @jobs = []
189       @cursor = {'a'=>nil,H=>Gdk::Cursor.new(Gdk::Cursor::SB_H_DOUBLE_ARROW),
190         V=>Gdk::Cursor.new(Gdk::Cursor::SB_V_DOUBLE_ARROW)}
191
192
193
194       set_events( Gdk::EventMotion::POINTER_MOTION_MASK
195           |Gdk::EventMotion::POINTER_MOTION_HINT_MASK)
196       signal_connect("expose_event") do
197         onExpose
198       end
199       signal_connect("motion_notify_event") do |w,e|
200         onMotion(e)
201       end
202       signal_connect("button_press_event") do |w,e|
203         onButtonPress(e)
204       end
205       signal_connect("button_release_event") do |w,e|
206         onButtonRelease(e)
207       end
208     end
209
210     def onExpose
211         alloc = self.allocation
212         checkPixbuf(alloc.width, alloc.height)
213         apixbuf = @pixbuf[@view]
214
215         gc = self.style.fg_gc(self.state)
216
217         oldfg = gc.foreground
218
219         if (apixbuf)
220           self.window.draw_pixbuf(gc, apixbuf,
221             0,0,0,0,-1,-1,Gdk::RGB::DITHER_NONE,0,0)
222
223           @fact[H] = alloc.width*1.0/@xres
224           @fact[V] = alloc.height*1.0/@yres
225
226           [H,V].each do |i| @wborders[i] = [@borders[i][0]*@fact[i],
227                           @borders[i][1]*@fact[i]] end
228
229           gc.set_line_attributes(2, Gdk::GC::LINE_SOLID,
230                           Gdk::GC::CAP_BUTT, Gdk::GC::JOIN_ROUND)
231           gc.rgb_fg_color = Gdk::Color.new(0x0000,0xFFFF,0x0000)
232
233           [0,1].each do |i|
234             self.window.draw_line(gc, @wborders[H][i], 0, @wborders[H][i],
235                           alloc.height)
236           end
237           gc.rgb_fg_color = Gdk::Color.new(0xFFFF,0xFFFF,0x0000)
238           [0,1].each do |i|
239             self.window.draw_line(gc, 0, @wborders[V][i], alloc.width,
240                   @wborders[V][i])
241           end
242
243           gc.set_line_attributes(1, Gdk::GC::LINE_SOLID, Gdk::GC::CAP_BUTT,
244                           Gdk::GC::JOIN_ROUND)
245           gc.rgb_fg_color = Gdk::Color.new(0xFFFF,0x2222,0x0000)
246           getLineStats unless (@imgData[@view]['xcutoff'])
```

```
247           [0,1].each do |i|
248             c = @imgData[@view]['xcutoff'][i]
249             if (c)
250             self.window.draw_line(gc, c * @fact[H], 0, c * @fact[H], alloc.height)
251             end
252           end
253
254           gc.set_line_attributes(3, Gdk::GC::LINE_SOLID, Gdk::GC::CAP_BUTT,
255                                  Gdk::GC::JOIN_ROUND)
256           gc.rgb_fg_color = Gdk::Color.new(0x0000,0x5555,0xFFFF)
257           avgData = @imgData[@view]['avgs']
258           avg_min = avgData[@borders[H][0]...@borders[H][1]].min
259           avg_max = avgData[@borders[H][0]...@borders[H][1]].max
260           avg_fact = alloc.height / (avg_max - avg_min)
261           x1 = 0
262           x2 = @fact[H]
263           y1 = alloc.height - (avgData[0] - avg_min) * avg_fact
264           (1...@xres).each do |x|
265             y2 = alloc.height - (avgData[x] - avg_min) * avg_fact
266             self.window.draw_line(gc, x1, y1, x2, y2)
267             x1 = x2
268             x2 += @fact[H]
269             y1 = y2
270           end
271
272
273
274         end
275
276         gc.foreground = oldfg
277         gc.set_line_attributes(1, Gdk::GC::LINE_SOLID, Gdk::GC::CAP_BUTT,
278                    Gdk::GC::JOIN_ROUND)
279         true
280       end
281
282       def onButtonPress(event)
283         if (event.button==1)
284           @lineDrag=@cursor['a']
285         end
286         true
287       end
288       def onButtonRelease(event)
289         if (event.button==1)
290           if (@lineDrag)
291             @lineDrag=nil
292             @fname.keys.each do |p| @imgData[p].delete('xcutoff') if (@imgData[p]) end
293             onExpose
294           end
295         end
296         true
297       end
298
299       def onRefresh
300             @realpixbuf= {}
301             @pixbuf= {}
302             onExpose
303       end
304
305       def onMotion(event)
306         return true unless(@pixbuf[@view])
307         if (@statusbar)
308           px = event.x/@fact[H]
309           exAVG = @imgData[@view]['avgs'][px]*(1 + px.to_i - px)
310                     + @imgData[@view]['avgs'][px+1]*(px - px.to_i)
311           @statusbar.push 1, sprintf("[%2d:%2d]    [%2.1f:%2.1f]",px.to_i,
312                     @imgData[@view]['avgs'][px],px,exAVG)
313         end
```

```
314       if (@lineDrag)
315         [V,H].each do |dir|
316           [0,1].each do |i|
317             if (@lineDrag==[dir,i])
318               @wborders[dir][i]= (dir==V) ? event.y : event.x
319               @borders[dir][i]=@wborders[dir][i]/@fact[dir]
320               onExpose
321             end
322           end
323         end
324       else
325         cursorChange = nil
326         [V,H].each do |dir|
327           @wborders[dir].each_index do |i|
328             pos = @wborders[dir][i]
329             if ((pos-LINESENS..pos+LINESENS) === ((dir==H) ? event.x : event.y))
330               cursorChange = [dir,i]
331             end
332           end
333         end
334
335         if (@cursor['a'])
336           if (cursorChange)
337             if (cursorChange!=@cursor['a'])
338               self.window.set_cursor(Gdk::Cursor.new(Gdk::Cursor::LAST_CURSOR))
339               self.window.set_cursor(@cursor[cursorChange[0]])
340               @cursor['a'] = cursorChange
341             end
342           else
343             self.window.set_cursor(Gdk::Cursor.new(Gdk::Cursor::LAST_CURSOR))
344             @cursor['a'] = nil
345           end
346         else
347           if (cursorChange)
348             self.window.set_cursor(@cursor[cursorChange[0]])
349             @cursor['a'] = cursorChange
350           end
351         end
352       end
353       true
354     end
355
356     def saveJobs(fname)
357       File.open(fname,"wb") do |outfile|
358         outfile.write(Marshal.dump(@jobs))
359       end
360     end
361     def loadJobs(fname)
362       puts "loading #{fname}"
363       infile = File.open(fname, 'rb')
364       @jobs = Marshal::load(infile.read)
365       infile.close
366       $csvoutname = fname
367     end
368
369     def readImages(fname)
370       dirname = File.dirname(fname)
371       @fname = getImageNames(fname)
372       @imgData = {}
373       @view = MIP
374       @imgData[@view] = readCSV
375       @view = AVG
376       @imgData[@view] = readCSV
377       @pixbuf = {}
378       @realpixbuf = {}
379       onExpose
380     end
```

```
381
382     def getAreas(data)
383       areas = Hash.new
384       ['softMipBlender','AvgMipBlender'].each do |proj|
385         points = data.select do |k,v| k=~/^#{proj}/ end.map do |d|
386           [d[1]['noise_norm'],d[1]['grad_norm']] end\
387           .select do |p| (p[0] > 0.0 && p[0] < 1.0) end\
388           .sort do |a,b| a[0]<=>b[0] end
389         points.push [1.0,1.0]
390
391         last = [0.0, 0.0]
392         area = 0
393         points.each do |p|
394           area += (p[0] - last[0]) * (p[1] + last[1]) /2.0
395           last = p
396         end
397         areas[proj] = area
398       end
399       areas
400     end
401
402
403     def storeCSV
404       data = {}
405       xvals = []
406       @fname.keys.each do |p|
407         @view = p
408         @imgData[@view] = readCSV unless (@imgData[@view])
409         getLineStats
410         data[p]={}
411         data[p]['x1']=@imgData[p]['xcutoff'][0]
412         data[p]['y1']=@imgData[p]['ycutoff'][0]
413         data[p]['x2']=@imgData[p]['xcutoff'][1]
414         data[p]['y2']=@imgData[p]['ycutoff'][1]
415         xvals.push [data[p]['x1'],data[p]['x2']].min.to_i
416         xvals.push [data[p]['x1'],data[p]['x2']].max.to_i+1
417       end
418       @fname.keys.each do |p|
419         avgs = @imgData[p]['avgs']
420         data[p]['std'] = getNoise(p,xvals.min,xvals.max)
421         data[p]['avgMin'] = avgs[xvals.min..xvals.max].min
422         data[p]['avgMax'] = avgs[xvals.min..xvals.max].max
423         data[p]['avgStd'] = avgs[xvals.min..xvals.max].std
424       end
425
426       grad_avg = (data[AVG]['y1']-data[AVG]['y2'])/(data[AVG]['x1']-data[AVG]['x2']).abs
427       grad_mip = (data[MIP]['y1']-data[MIP]['y2'])/(data[MIP]['x1']-data[MIP]['x2']).abs
428       noise_avg = data[AVG]['std']
429       noise_mip = data[MIP]['std']
430       grad_span = grad_mip - grad_avg
431       noise_span = noise_mip - noise_avg
432
433       @fname.keys.each do |p|
434         grad = (data[p]['y1']-data[p]['y2'])/(data[p]['x1']-data[p]['x2']).abs
435         data[p]['grad_norm'] = (grad - grad_avg) / grad_span
436         data[p]['noise_norm'] = (data[p]['std'] - noise_avg) / noise_span
437       end
438
439       areas = getAreas(data)
440
441       oldFile = File.exist?($csvoutname + "_data.csv")
442       File.open($csvoutname + "_data.csv","a+") do |outfile|
443         CSV::Writer.generate(outfile) do |csv|
444           if (!oldFile)
445             line = @imgData[MIP]['meta'].select do |x| x[0]!='Projection' end\
446                   .collect do |d| d[0] end
447             line.push "mipFilename"
```

```
448             ['h','v'].each do |d|
449               [1,2].each do |n|
450                 line.push "border_#{d}#{n}"
451               end
452             end
453             @fname.keys.sort.each do |p| line.push p+"_noise_norm" end
454             @fname.keys.sort.each do |p| line.push p+"_grad_norm" end
455             @fname.keys.sort.each do |p| line.push p+"_imagequality" end
456             areas.keys.sort.each do |ak| line.push ak+"_area" end
457             csv << line
458           end
459           line = @imgData[MIP]['meta'].select do |x| x[0]!='Projection' end\
460                   .collect do |d| d[1] end
461           line.push @fname[MIP]
462           [H,V].each do |d|
463             @borders[d].each do |b|
464               line.push b
465             end
466           end
467           @fname.keys.sort.each do |p| line.push data[p]['noise_norm'] end
468           @fname.keys.sort.each do |p| line.push data[p]['grad_norm'] end
469           @fname.keys.sort.each do |p|
470             line.push Math.sqrt( (1.0 - data[p]['grad_norm']) ** 2
471                   + data[p]['noise_norm'] ** 2)
472           end
473           areas.keys.sort.each do |ak| line.push areas[ak] end
474           csv << line
475         end
476       end
477       oldFile = File.exist?($csvoutname + "_rawData.csv")
478       File.open($csvoutname + "_rawData.csv","a+") do |outfile|
479         CSV::Writer.generate(outfile) do |csv|
480           if (!oldFile)
481             line = @imgData[MIP]['meta'].select do |x| x[0]!='Projection' end\
482                   .collect do |d| d[0] end
483             line.push "mipFilename"
484             ['h','v'].each do |d|
485               [1,2].each do |n|
486                 line.push "border_#{d}#{n}"
487               end
488             end
489             @fname.keys.sort.each do |p|
490               %W(Projection x1 y1 x2 y2 std avgMin avgMax avgStd).each do |c|
491                 line.push p+"_"+c
492               end
493             end
494             csv << line
495           end
496           line = @imgData[MIP]['meta'].select do |x| x[0]!='Projection' end\
497                         .collect do |d| d[1] end
498           line.push @fname[MIP]
499           [H,V].each do |d|
500             @borders[d].each do |b|
501               line.push b
502             end
503           end
504           @fname.keys.sort.each do |p|
505               line.push p
506               %W(x1 y1 x2 y2 std avgMin avgMax avgStd).each do |c|
507                 line.push data[p][c]
508               end
509           end
510           csv << line
511         end
512       end
513       puts "stored data to csv"
514     end
```

```
515
516     def showProjection(p)
517       @view = p
518       @imgData[@view] = readCSV unless (@imgData[@view])
519       onExpose
520     end
521
522     def addJob
523       job = {'borders'=>@borders.dclone,'fname'=>@fname[MIP].clone}
524       @jobs.push job
525       puts "job added"
526     end
527
528     def executeJobs
529       @jobs.each do |j|
530         if (!(j['fname']==@fname))
531           @imgData = {}
532           @pixbuf = {}
533           @realpixbuf = {}
534         end
535         @fname = getImageNames(j['fname'])
536         @borders = j['borders']
537         storeCSV
538       end
539       puts "finished all jobs"
540       @jobs = []
541     end
542
543
544   private
545
546     def getImageNames(fname)
547       dirname = File.dirname(fname)
548       fileID = File.basename(fname).gsub(/^[^_]+/, '')
549       imageNames = {}
550       dir = Dir.new(dirname)
551       dir.select do |d| d.match("#{fileID}$") end.map do |n| n.gsub(/_.*$/,'') end\
552           .each  do |x|
553                   imageNames[x] =  dirname + File::SEPARATOR + x + fileID
554           end
555       imageNames
556     end
557
558
559
560     def readCSV
561       if (@fname[@view])
562         csvreader = CSV::Reader.create(File.open(@fname[@view], 'rb'))
563         data = {}
564         data['lines'] = []
565         data['avgs'] = []
566         data['meta'] = []
567         i = 0
568         csvreader.each do |row|
569           if (i<10)
570             data['meta'].push row
571             i+=1
572           else
573             line = row.collect do |v| v.to_i end
574             data['lines'].push line
575           end
576         end
577         csvreader.close
578         if (@xres!=data['lines'].length || @yres!=data['lines'][0].length)
579           @yres = data['lines'][0].length
580           @xres = data['lines'].length
581           wrong_borders = nil
```

```
582           [0,1].each do |i|
583             if (@borders[i])
584               wrong_borders = 1 if (@borders[H][i] > @xres || @borders[H][i] < 0)
585               wrong_borders = 1 if (@borders[V][i] > @yres || @borders[V][i] < 0)
586             else
587               wrong_borders = 1
588             end
589           end
590           if (wrong_borders)
591             @borders[H] = [(0.05 * @xres).to_i,(@xres-0.05 * @xres).to_i]
592             @borders[V] = [(0.05 * @yres).to_i,(@yres-0.05 * @yres).to_i]
593           end
594         end
595         data
596       end
597     end
598
599
600
601     def getNoise(p,x1,x2)
602       statbuffer = []
603       @imgData[p]['lines'][(x1..x2)].each do |line|
604         lineScope = line[@borders[V].min...@borders[V].max]
605         lsavg = lineScope.avg
606         statbuffer.concat(lineScope.collect do |v| v-lsavg end)
607       end
608       statbuffer.std
609     end
610
611     def getLineStats
612       avgs = (0...@xres).collect do |x|
613         @imgData[@view]['lines'][x][@borders[V].min...@borders[V].max].avg
614       end
615
616       hBorderMin = @borders[H].min.to_i
617       hBorderMax = @borders[H].max.to_i
618       @imgData[@view]['avgs'] = avgs
619       scopeMaxInd = hBorderMin
620       scopeMinInd = hBorderMin
621       for x in hBorderMin..hBorderMax
622         scopeMaxInd = x if (avgs[x]>avgs[scopeMaxInd])
623         scopeMinInd = x if (avgs[x]<avgs[scopeMinInd])
624       end
625       scope = [scopeMinInd,scopeMaxInd]
626       scopeRange = avgs[scopeMaxInd] - avgs[scopeMinInd]
627
628       xval = [scope.min,scope.max]
629       cutoff = [avgs[scopeMinInd] + LOWCUTOFF * scopeRange,
630           avgs[scopeMinInd] + HIGHCUTOFF * scopeRange]
631       adder = 1
632       muller = 1
633       order = [0,1]
634       if (scopeMaxInd<scopeMinInd)
635         adder = -1
636         order = [1,0]
637       end
638       @imgData[@view]['xcutoff'] = [nil,nil]
639       @imgData[@view]['ycutoff'] = cutoff
640       order.each do |v|
641         xcut = nil
642         while ((scope.min..scope.max) === xval[v])
643           if ( (avgs[xval[v]]*muller) > (cutoff[v]*muller) )
644             xcut = xval[v] unless (xcut)
645           else
646             xcut = nil
647           end
648           xval[v] +=adder
```

```
649         end
650         if (xcut)
651           low = avgs[xcut-adder]
652           high = avgs[xcut]
653           f = (cutoff[v]-low)/(high-low)
654           @imgData[@view]['xcutoff'][v] = xcut + (f-1) * adder
655         end
656         adder = -adder
657         muller = -muller
658       end
659     end
660
661     def getStats
662       getLineStats
663       statbuffer = []
664       @imgData[@view]['lines'][(@borders[H].min...@borders[H].max)].each do |line|
665         statbuffer.concat(line[@borders[V].min...@borders[V].max])
666       end
667       {'avg' => statbuffer.avg, 'std' => statbuffer.std}
668     end
669
670     def checkPixbuf (w,h)
671       @pixbuf[@view] = getPixbuf(w,h) unless (@pixbuf[@view] &&
672         @pixbuf[@view].width==w && @pixbuf[@view].height==h)
673     end
674
675     def getPixbuf (width,height)
676       if (! @realpixbuf[@view] )
677         aData = @imgData[@view]
678         imgData = Array.new(width*height*3,100)
679         if (aData)
680           aData = aData['lines']
681           myStats = getStats
682           low  = myStats['avg']-myStats['std'];
683           fact = 255/(myStats['std']*2);
684           index = 0
685           for y in 0...@yres
686             for  x in 0...@xres
687                 col = ((aData[x][y]-low)*fact).to_i;
688                 col = 0 if col<0
689                 col = 255 if col>255
690                 imgData[index] = col
691                 index +=1
692                 imgData[index] = col
693                 index +=1
694                 imgData[index] = col
695                 index +=1
696             end
697           end
698           dat = imgData.pack('C*')
699           realPixBuf = Gdk::Pixbuf.new(dat, Gdk::Pixbuf::COLORSPACE_RGB,
700             false, 8, @xres, @yres, @xres*3)
701           @realpixbuf[@view]=realPixBuf
702         end
703       end
704       if ( @realpixbuf[@view] )
705         @pixbuf[@view] = @realpixbuf[@view].scale(width, height,
706                   Gdk::Pixbuf::INTERP_BILINEAR)
707       end
708     end
709   end
710
711   main
```

Anhang C

StoneSorter

```
1
2  require 'csv'
3
4  KIDNEY = 0
5  URETER = 1
6
7  TP = 0
8  FP = 1
9  FN = 2
10 TN = 3
11 ORGSTR = []
12 ORGSTR[KIDNEY] = "Niere"
13 ORGSTR[URETER] = "Ureter"
14
15 def main
16   patients = Array.new
17   fname = ARGV.shift.dclone
18   csvreader = CSV::Reader.create(File.open(fname, 'rb'))
19   csvreader.each do |row|
20       patients.push Patient.new(row)
21   end
22   csvreader.close
23
24
25   fname.sub!(/\.csv$/,'')
26
27   ['opt','noopt'].each do |opt|
28     [KIDNEY,URETER].each do |organ|
29       ['Ref1','Ref2'].each do |ref|
30         File.open(fname + ".#{ORGSTR[organ]}.#{ref}.#{opt}.csv","w+") do |outf|
31           File.open(fname + ".#{ORGSTR[organ]}.#{ref}.#{opt}.dis.csv","w+") do |disf|
32             File.open(fname + ".#{ORGSTR[organ]}.#{ref}.#{opt}.nondis.csv","w+") do |nondisf|
33               CSV::Writer.generate(outf) do |csv|
34                 CSV::Writer.generate(disf) do |discsv|
35                   CSV::Writer.generate(nondisf) do |nondiscsv|
36                     csv << %W(TP FP FN TN)
37                     csv << [0,1,2,3]
38                     csv << %W(Name softMip MipAvg)
39                     patients.each do |p|
40                       p.putOrgan(csv,discsv,nondiscsv,organ,ref,opt)
41                     end
42                   end
43                 end
44               end
45             end
```

```
46              end
47            end
48          end
49        end
50      end
51    end
52
53    class Object
54      def dclone
55        Marshal::load(Marshal.dump(self))
56      end
57    end
58
59    class Optimizer
60      attr_reader :versions
61
62      def initialize(set,ref)
63        @set = set.dclone
64        @ref = ref
65        @flatref = ref.flatten
66        @current = [[0,0,0],[0,0,0],[0,0,0],[0,0,0]]
67        @versions = []
68        @minDiff = 10000
69      end
70
71      def fillCurrent(size=0,loc=0)
72        while (loc<3 && @set[size][loc]==0)
73          size+=1
74          if (size==4)
75            size = 0
76            loc+=1
77          end
78        end
79        if (loc<3)
80          @set[size][loc]-=1
81          ppos = getPossNewLoc(size,loc)
82          ppos.each do |p|
83            @current[p[0]][p[1]]+=1
84            fillCurrent(size,loc)
85            @current[p[0]][p[1]]-=1
86          end
87          @set[size][loc]+=1
88        else
89          diff = getDiff()
90          if (diff < @minDiff)
91            @versions = []
92            @minDiff = diff
93          end
94          if (diff == @minDiff)
95            if (!@versions.include? @current)
96              @versions.push @current.dclone
97            end
98          end
99        end
100     end
101
102     def getPossNewLoc(size,loc)
103       pnloc = []
104       (-1..1).each do |dsize|
105         (-1..1).each do |dloc|
106           nsize = size + dsize
107           nloc = loc + dloc
108           if (nsize>=0 && nloc>=0 && nsize<4 && nloc<3)
109               pnloc.push [nsize,nloc]
110           end
111         end
112       end
```

```
113       if (pnloc.size==0)
114         pnloc.push [size,loc]
115       end
116       pnloc
117     end
118
119     def getDiff
120       fcurr = @current.flatten
121       diff = 0
122       fcurr.each_index do |x| diff += (fcurr[x] - @flatref[x]).abs end
123       diff
124     end
125
126   end
127
128
129   class Patient
130     attr_reader :name, :stones
131     def initialize(row)
132       @stones = Hash.new
133       @name = row.shift
134       @stones['softMip'] = readStones(row)
135       @stones['MipAvg'] = readStones(row)
136       @stones['Ref1'] = readStones(row)
137       @stones['Ref2'] = readStones(row)
138     end
139
140     def putOrgan(csv,discsv,nondiscsv,organ,ref,opt)
141       r = Array.new
142       [0,1].each do |side|
143         if (opt == 'opt')
144           sets = getSets(side,organ,ref)
145           r.concat(rateSets(sets['softMip'],sets['MipAvg'],@stones[ref][side][organ]))
146         else
147           r.concat(rateSets(@stones['softMip'][side][organ],
148                   @stones['MipAvg'][side][organ],@stones[ref][side][organ]))
149         end
150       end
151       if (r.size==0)
152         r.push [TN,TN]
153       end
154       r.each do |l|
155         line = [@name]
156         line.push l[0]
157         line.push l[1]
158         csv << line
159         if (l[0] == TP || l[0] == FN) # diseased
160           discsv << line
161         else
162           nondiscsv << line
163         end
164
165       end
166     end
167
168     def getSets(side,organ,ref)
169       softMipO = Optimizer.new(@stones['softMip'][side][organ],@stones[ref][side][organ])
170       mipAvgO = Optimizer.new(@stones['MipAvg'][side][organ],@stones[ref][side][organ])
171       softMipO.fillCurrent
172       mipAvgO.fillCurrent
173
174       smV = softMipO.versions[0]
175       maV = mipAvgO.versions[0]
176       minDiff = getSetDiff(smV,maV)
177
178       softMipO.versions.each do |sm_v|
179         mipAvgO.versions.each do |ma_v|
```

```
180           diff = getSetDiff(sm_v,ma_v)
181           if (diff < minDiff)
182             smV = sm_v
183             maV = ma_v
184              minDiff = diff
185           end
186         end
187       end
188       {'softMip'=>smV, 'MipAvg'=>maV}
189     end
190
191     def rateSets(set1,set2,ref)
192       rates = Array.new
193       (0..3).each do |size|
194         (0..2).each do |loc|
195           stones = [set1[size][loc],set2[size][loc],ref[size][loc]].max
196           (1..stones).each do |s|
197             r1,r2 = nil,nil
198             if (ref[size][loc]>=s) # stone exists
199               if (set1[size][loc]>=s) # stone detected
200                 r1 = TP
201               else
202                 r1 = FN
203               end
204               if (set2[size][loc]>=s) # stone detected
205                 r2 = TP
206               else
207                 r2 = FN
208               end
209             else # stone does not exist
210               if (set1[size][loc]>=s) # stone detected
211                 r1 = FP
212               else
213                 r1 = TN
214               end
215               if (set2[size][loc]>=s) # stone detected
216                 r2 = FP
217               else
218                 r2 = TN
219               end
220             end
221             rates.push [r1,r2]
222           end
223         end
224       end
225       rates
226     end
227
228     def getSetDiff(a,b)
229       af = a.flatten
230       bf = b.flatten
231       diff = 0
232       af.each_index do |x| diff += (af[x] - bf[x]).abs end
233       diff
234     end
235
236
237
238     def readStones(row)
239       stones = Array.new
240       2.times do
241         stones.push readSide(row)
242         2.times do row.shift end
243       end
244       stones
245     end
246     def readSide(row)
```

```
247       side = Array.new
248       side[KIDNEY] = Array.new
249       side[URETER] = Array.new
250       4.times do
251         side[KIDNEY].push readLoc(row)
252         side[URETER].push readLoc(row)
253         row.shift
254       end
255       side
256     end
257     def readLoc(row)
258       loc = Array.new
259       3.times do loc.push row.shift.to_i end
260       loc
261     end
262   end
263
264
265   main
```

Lebenslauf

persönliche Daten

Name	Henning Meyer
Geburt	am 1.5.1978 in Berlin
Familienstand	ledig
Nationalität	deutsch

Schul-/Hochschulbildung

1997	Erlangung der allgemeinen Hochschulreife
10/1998-12/2004	Studium der Humanmedizin an der Medizinischen Fakultät der Humboldt-Universität zu Berlin (Charité)
09/2000	Physikum
08/2001	1. Staatsexamen
09/2003	2. Staatsexamen
12/2004	3. Staatsexamen
seit 07/2005	Anstellung als wissensschaftlicher Mitarbeiter am Institut für Radiologie der Charité – Universitätsmedizin Berlin

Selbständigkeitserklärung

Ich, Henning Meyer, erkläre, dass ich die vorgelegte Dissertationsschrift mit dem Thema: “softMip – Entwicklung eines neuen Projektionsverfahrens für die digitale Schnittbildgebung und Evaluation anhand von Ultra-Low-Dose-CT-Aufnahmen zur Harnwegskonkrementsuche” selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, ohne die (unzulässige) Hilfe Dritter verfasst und auch in Teilen keine Kopien anderer Arbeiten dargestellt habe.

Henning Meyer

10. Dezember 2005